



University of North Dakota
UND Scholarly Commons

Theses and Dissertations

Theses, Dissertations, and Senior Projects

January 2012

A Comparative Analysis To Validate The Benefits Of Formal Versus Informal Software Model Transformation

Kaden Daley

Follow this and additional works at: <https://commons.und.edu/theses>

Recommended Citation

Daley, Kaden, "A Comparative Analysis To Validate The Benefits Of Formal Versus Informal Software Model Transformation" (2012). *Theses and Dissertations*. 1281.
<https://commons.und.edu/theses/1281>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Senior Projects at UND Scholarly Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UND Scholarly Commons. For more information, please contact zeinebyousif@library.und.edu.

A COMPARATIVE ANALYSIS TO VALIDATE THE BENEFITS OF FORMAL
VERSUS INFORMAL SOFTWARE MODEL TRANSFORMATION

by

Kaden Daley

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

In partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

August


2012

Copyright 2012 Kaden Daley

This thesis, submitted by Kaden Daley in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and hereby approved.



Chairperson



This thesis meets the standards for appearance, conforms to the style and format requirements of the Graduate School of the University of North Dakota, and hereby approved.



Dean of the Graduate School



Date

Title A Comparative Analysis to Validate the Benefits of Formal Versus Informal Software Model Transformation

Department Computer Science

Degree Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Kaden Daley
July 25, 2012

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	ix
ACKNOWLEDGEMENTS	xi
ABSTRACT	xiii
CHAPTER 1	1
INTRODUCTION	1
1.1 Problem Definition	1
1.2 Significance of Work	2
1.3 Literature Review	3
1.4 Problem Domain	4
1.5 Scope of Work	5
1.6 Expected Results	6
1.7 Structure of Research	6
CHAPTER 2	7
BACKGROUND	7
2.1 UML	7
2.2 Class Diagram	9

2.3 Database	11
2.3.1 Normal Form.....	13
2.3.2 SQL	15
2.4 Model Transformation	16
2.5 Related Work	17
2.5.1 Related Work 1	17
2.5.2 Related Work 2	18
2.5.3 Related Work 3	19
2.5.4 Related Work 4	21
CHAPTER 3	23
TRANSFORMATION METHODOLOGY.....	23
3.1 Description of Methodologies.....	23
3.1.1 Methodology I.....	23
3.1.2 Methodology2	28
3.2 Case Study Class Diagram.....	28
3.3 Applying Methodologies	30
3.3.1 Methodology 1	30
3.3.2 Methodology2	32
3.4 Resulting Schemas	53
CHAPTER 4	58

ANALYSIS.....	58
4.1 Methodology 1	58
4.2 Methodology 2	61
4.3 Comparison of Methodologies.....	63
4.4 Evaluation of Class Diagrams.....	64
4.5 Evaluation of Methodologies.....	65
4.6 Similarity and Difference of KFP and AFR Methodologies	66
4.6.1 Methodology 1	66
4.6.2 Methodology 2	67
4.7 Comparison of Conclusions.....	68
CHAPTER 5	70
CONCLUSION.....	70
5.1: Validation Process	70
5.2 Future Work	71
Appendix A.....	73
Appendix B.....	75
Appendix C.....	77
Appendix D.....	87
REFERENCES.....	95

LIST OF FIGURES

Figure	Page
1. Class diagram	9
2. Classification of Relationship	11
3. Functional Dependency.....	14
4. Class diagram of Kalman Filter Program	26
5. Extended Class Diagram of Kalman Filter Program	27
6. Class diagram and extended class diagram of KFP	31-32
7. Hyper graph of relations in the case study	33-34
8. Graham Reduction process of KFP	34-45
9. Join tree of KFP.....	46
10. Partial Table structure using Methodology I	61

LIST OF TABLES

Table	Page
1. Four- layer metamodel hierarchy	8
2. Types of Cardinalities.....	10
3. Normal Forms.....	13
4. Quantitative comparison of AFR.....	21
5. Stereotypes for database design	23
6. UML extension	24
7. Guidelines for object-relational database design.....	26
8. Stereotyped class of KFP.....	29
9. Stereotyped attributes of Kalman Filter Program.....	30
10. Abbreviation of stereotyped classes of KFP.....	33
11. Application of Algorithm B	47
12. SQL:2003 schema drawn using Methodology I	54-55
13. SQL: 2003 schema drawn using Methodology 2	56-57
14. Oracle 11g SQL syntax.....	58
15. Methodology1 Oracle 11g SQL schema.....	59
16. Abbreviation of Class Name	60-61
17. Methodology2 Oracle 11g SQL schema	62
18. Kalman_Filter	63

19. Quantitative comparison	64
20. Comparison of Methodology 1	67
21. Comparison of Methodology 2	68

ACKNOWLEDGEMENTS

I would like to thank my advisory committee for all their support to fulfill the necessary requirements. I would like to thank the staff at the Computer Science Department for giving me all the support needed to reach my goal. Thanks to Washington Helps and Micheal Crawford for assisting in getting this document in the necessary format.

Jeneen and Naomi Daley, I thank you both for the support, encouragement and time which allowed me to pursue this Master's Degree. Also I want to thank Dr. Grant and Bishop M. Cole for their support.

To: Jeneen and Naomi

ABSTRACT

In object –oriented development the Unified Modeling Language (UML) is the ISO/IEC standard for modeling language and is supported by major corporations. In relational database development, entity-relationship models have traditionally been use for modeling such systems. Transforming from one notation to another notation is of great importance in developmental environments where this is required. There are several techniques for transforming UML models to object-relational database systems. Prior assessment has been carried out on transforming UML class diagram models to object-oriented relational databases, which yield significant results. One approach to transformation may involve the use of formal (mathematical) techniques, while other approaches may rely on informal techniques to accomplish the transformation. The use of a formal technique to transform may incorporate graph-theory on UML class diagram. An informal technique may be utilized in transforming UML extension mechanisms, to represent object-relational concepts.

A prior research effort examined the benefits of two such approaches in transforming UML class diagram models into object-relational database representation. That work, sought to determine the benefits of one approach (formal technique) versus the benefits of the other approach (informal technique), by way of comparative analysis. The researchers drew inferences from the comparative analysis as to the suitability of one approach versus the other on classes of problem domains. The results of such work have to be validated in order for it to become acceptable and its implication applied in software development decision making. In this work there will be an attempt to apply a similar

comparative analysis on a model from a different application domain (Kalman Filter Program Representation), from that which was used in the first study an (Airline Flight Reservation System). The goal of this research is to provide validation of the usefulness of this type of comparative analysis.

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

As scientists invent new methods to solve or to assist humans in solving their daily problems, it is required to have independent researcher modeling methods outlined to verify prior work. Preliminary research draws conclusion that have significant implications to future work in specific research areas. For scientific work to be accepted by any community of researchers or applied to software development, they will have to be validated.

A comparative analysis of two approaches to transform object-oriented design models to object relational database representations was conducted previously by Chennamaneni's in her published papers [19, 21] and thesis [7]. Two transformation approaches were applied, namely formal and informal transformation [14, 15]. Conclusions of the two approaches were made as well as recommendation about the suitability of either approach to a particular domain. Within the scientific community, for these recommendations to be accepted further research has to be persued in the form of repeating the method taken in reaching those recommendations and applying it to a different problem domain.

The work documented in this thesis aims to validate prior work done by R. Chennamaneni [7], by applying the two approaches for transforming UML class diagram models to object-relational database systems to a different problem domain, aimed at reaching similar conclusions. This thesis work is centered on one of the recommendations made by Chennamaneni which seeks for validation of her conclusions before acceptance of her recommendation.

The validation process will involve the use of the conclusions reached from this research, from applying the formal and the informal approaches and compare it with, the conclusions reached by Chennamaneni's work.

1.2 Significance of Work

Applying technology is inevitable in making humans life easier in carrying out daily task and solving problems. Researchers try to envision creative means to empower humans in doing their daily work. With the increase use of database to store information, scientists have to come up with efficient ways of using database to serve the desired purpose. With existing practices and tools available, it is vital to develop new techniques that are compatible to existing systems.

UML is the accepted standard by ISO for modeling design process in software industries [5]. With UML being a standard in software industry, researchers need to develop modeling software approaches to match software development process.

Verifying work done in this area by R. Chennamaneni [7] will prove that the methodologies proposed can be applied to other problem domains. Paving the way with

the static modeling verification, researchers will be able to apply this modeling to dynamic verification and validation.

Other benefits from this research include the automating areas of the transformation process. Automation may be applied where derivation of SQL statements from extended class diagram exist, also in extension of the class diagram from the basic class diagram. This would reduce the time taken for transforming UML diagrams into SQL statements. This research will also foster researchers with the knowledge regarding which methodology is best suited for a particular problem domain in regards of, the space required to create tables or the time taken to carry out the transformation process.

1.3 Literature Review

One of the primary resources of this research is from the work of R. Chennamaneni [7], which carried out a static analytical research of two methodologies. The two methodologies are the formal and the informal transformation of UML class diagrams into their respective SQL representations. The informal methodology applies UML extension mechanism on the class diagrams that derives stereotyped equivalent class diagrams, which are then transformed into SQL statements. The approach taken by the formal methodology is to generate nested tables from the class diagrams by applying two graph theory based algorithms. From these nested tables SQL statements are developed.

The researcher applied these two methodologies to a Flight Reservation System (FRS), from which a quantitative comparison was carried out on the schema elements.

Analyzing the results, the researcher was able to make conclusions about the two methodologies applied. Some conclusions reached were: formal methodology requires intimate knowledge of mathematics, while informal methodology relies on UML knowledge. Other conclusion includes informal methodology required fewer processing steps while informal required more processing steps.

Another important work to this research is that of E. Marcos, et al [14] which focused on the use of UML to model an entire system as opposed to E/R diagrams which focused on one model. With UML extendable capability it allows the introduction of stereotypes for various applications. The author focused on the introduction of some guidelines for transforming UML conceptual schema into object oriented-relational schema. This informal methodology provides the steps needed to transform the Kalman Filter Program UML class diagram into SQL statements.

The work of Wai and David [15], provide the framework for the formal methodology. Their work look at how to derive NNF nested tables from class diagrams which forms SQL:1999 statements. Two algorithms are applied to this process to yield the result of SQL statements.

1.4 Problem Domain

As we take an outer look at our planet earth, we see man-made objects orbiting it. A closer look at how these object such as spacecraft, geographical system, or even how an aircraft works, they apply some form of Kalman Filter algorithm [13]. Gyroscopes are used to measure altitude but it degrades over time, so Kalman Filter provides optimal

estimates which assist object staying on the desired course. This recursive algorithm provides estimation of a state such as its position, altitude or even noise characteristics.

An UML class diagram was developed for the Kalman Filtering domain, which shows the static concepts which entails process model and measurement model. Process model consist of aggregations such as state, process or control while measurement models can be either linear or nonlinear. For this problem domain a class diagram is shown in Figure 4 of a Kalman Filter Program [13].

This researcher will use the UML class diagram from Kalman Filter Program domain: transforming it into object-relational database system.

1.5 Scope of Work

This research will be a static analysis of transforming UML class diagram from the Kalman Filter Program class diagram into SQL statement. Two methodologies will be applied to this domain, first an informal methodology which heavily relies on UML concepts. The second methodology is a formal methodology which is based on mathematical principles. From the results of these two methodologies, the derivation of SQL statements, which will be compared based on schema elements looking for similarities or differences, as the prior research was conducted. An overall comparison of this work will be conducted with the prior work from another problem domain.

1.6 Expected Results

The results anticipated from this research should yield similar results to the prior research that was carried out by Chennamaneni [7]. The first expecting result should derive SQL statements from using this new problem domain. The informal methodology SQL statements should produce more tables with simple structure while the formal methodology should produce fewer tables with complex structure.

Having these result noted above, will enable a comparison of similarities with the work of Chennamaneni [7], thus verifying her work. However, if the result is different from the previous work, further investigation will be needed to examine the transformation techniques, whether they were applied correctly. Another conclusion which may be drawn from variant results is, not all problem domains are able to transform correctly into an object-relational database system.

1.7 Structure of Research

The rest of the thesis is organized as follows: Chapter 2 will present related terminologies and background related work which is essential to this work. Chapter 3 will provide the application of both methodologies to our case study. Chapter 4 will present an analysis of the two methodologies. In Chapter 5, a comparison of the results from the first case study and this present research case study will be discussed as well as future work.

CHAPTER 2

BACKGROUND

In order to present any scientific based research, the terminologies associated with this work, needs to be established. The first step will be looking at terms associated with UML, followed by terms associated with database, then finally significant research papers that contribute to this field of research.

2.1 UML

Unified Modeling Language (UML)¹, is a semiformal graphical language commonly used in software industries to model design process, structure, relationship and interaction among components or elements of systems. The current version of UML is 2.4 as of March 2011², which has limited documentation available and currently not the acceptable international standard. References will be made from version 1.4.2 which is an ISO [5] standard.

Structural and behavioral modeling characterizes the two general sets of UML models. The structural aspect of UML focuses on the static architecture of a model, while the behavioral focus on interaction of a system otherwise known as the dynamic aspect.

¹ www.uml.org

² www.omg.org

Some of the diagrams that are available to UML includes: package, class, object, use case, component, deployment and interaction diagrams. UML 2.3³ clearly defines thirteen (13) diagrams, updates over the years, introduces new diagrams, such as timing and constraints, while features from some of the diagrams have been revised.

UML architecture is designed around a four layer meta-model structure⁴, which is illustrated in Table 1, where M0, M1, M2, M3 refers to the layers: user objects, model, metamodel, and meta-metamodel levels respectively. The meta-modeling provides the foundation to define the language for this architecture at a high level of abstraction. At the metamodel level it defines a model for specifying a model. It provides more details than a meta-metamodel, such as attributes, class, operation and component.

Table 1: Four- layer metamodel hierarchy

Layer	Description
Meta-metamodel	Establish a language for specifying metamodels.
Metamodel	An instance of a meta-metamodel
Model	An instance of a metamodel
User objects	An instance of a model

The model level provides an instance of metamodel. The aim of this layer is to define a language that describes an information or problem domain. User objects layer

³ www.sparxsystems.com

⁴ <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/>

describes specific information or problem domain. UML architecture is in no way limited to four meta –layers, a user may define additional layers, to define the domain they are modeling.

Based on the various diagrams within UML, the class diagram will be exploited based on the scope of this work which looks at the interrelationship among classes. A detailed description of a class diagram is described in the following section.

2.2 Class Diagram

A class diagram is used to show the static structure of the system, such as the interrelationships among classes or the relationship with other model elements. Figure 1 shows an example of a class diagram which is represented by a rectangular box consisting of three components: class name, attributes and operations. Names for a class diagram are normally given based on the object being modeled. Attributes in a class diagram is the middle section which describes values, while an operation is a function or procedure that a class can carry out which is represented in the bottom section of the rectangle box [2].

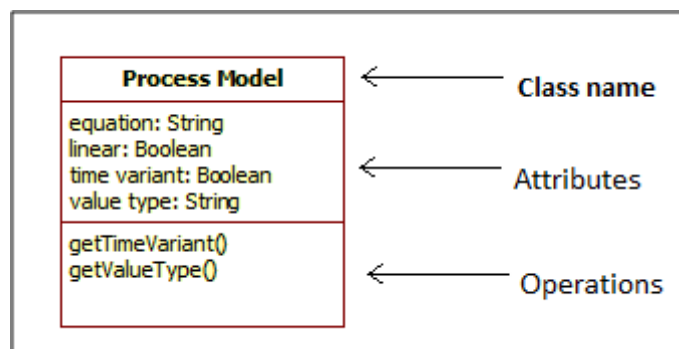


Figure 1: Class diagram

Constraints, stereotypes may be used to define classes. Constraints are considered as the rule about a data field, e.g. <<PK>> or <<FK>>, i.e. primary key or foreign key. Stereotype is a specialized UML element, which allows the user to extend the semantics of UML element to suit desired needs of the domain. [3] Class diagrams are capable of modeling objects from the real world, with a detail description or a high level view of the objects. To construct a class diagram of an object, involves the drawing of a box and enter the class name. Any association between other classes, a line is used to connect these classes. The cardinality between classes are showed in Table 2 , where a designer can specify the number of objects such as 1 to 1 relationship may exist between two classes or other multiplicity of relationship.

Table 2: Types of Cardinalities

Cardinality	Meaning
0..*	Zero or more
0..1	Zero or one
1..*	One or more
1	One only
n	n only (n > 1)
0..n	Zero to n (n > 1)
1..n	One to n (n > 1)

A cardinality with 0..* means a relationship of zero or more elements, where * is an unbound number. The same relationship exists with a 0..n, except the n being a specified upper bound number. Cardinalities with 1..* or 1..n suggest the relationship consist of at least 1 in the lower bound relationship. The relationship of 0..1 means one relationship or a case of zero relationship may exist.

For the designing of a database, a detailed class diagram would provide the semantics which includes the data entities, their association and relationship among classes [1]. Figure 2 shows some classification of UML relationships, where association shows relationship between two or more elements. Aggregation is considered as a special case of association, showing relationship between a whole and its parts. Classes in aggregation are at the same level represented with an open diamond at the whole-end as seen in Figure 2. Composition is a strong form of aggregation and Generalization shows relationship between a super and a sub-class [4,16].

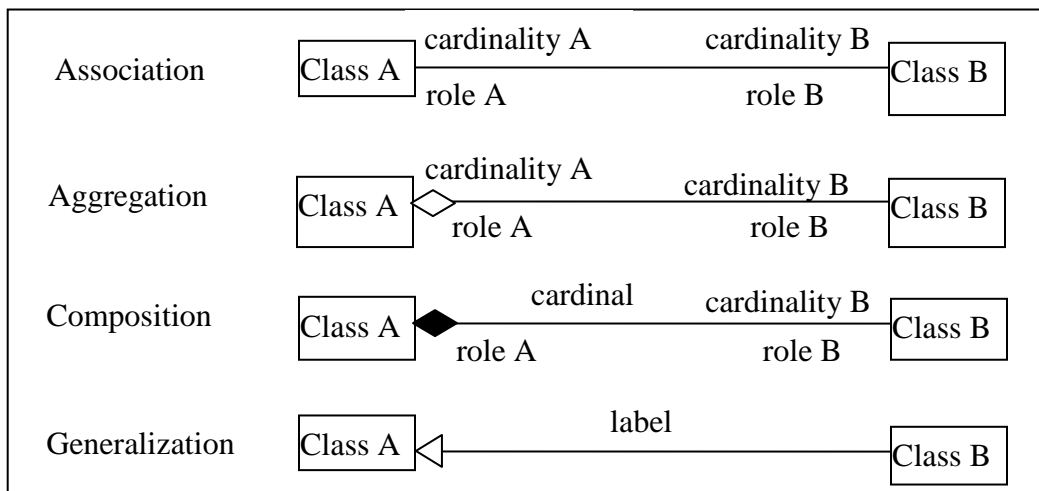


Figure 2: Classification of Relationship

2.3 Database

A database can be considered as a collection of related elements. The name, address, or telephone number of an individual, can be considered as data. Once these data have been processed into useful form it is considered as information. One way of looking

at a database is to look at the file component which includes data at the lower level which makes up a field. A group of fields makes up a record, while a collection of records constitute a file.

The representation of data structure and relationships within a database is known as database model. Conceptual and implementation models are two categories of database model. Conceptual database model focuses on what is represented in the database, e.g. Entity Relation (E-R) [4]. One to many, many to one, and one to one are the only cardinality used in conceptual database models to describe database relationships which are not flexible as UML features in Table 2. How the data is structure is the focus of the implementation model, e.g. hierarchical, network, relational database and object- oriented database model [9].

Relational databases stores data within a single structure, which is represented in a table format in rows and columns. A primary key and a unique column name are required to represent data in a relational database. A file is referred as a relation, record as a tuple and field as an attribute in relational database. Access, MySQL, Sybase, SQL Server are some examples of relational databases. [10]

Object-Oriented Databases (OODB) contains data and instruction on how to process the data is stored in the objects. In addition to a person name, address, telephone or address, the object feature allows picture, audio or video to be stored [10]. This architecture design allows real live data to be represented. Object Relational- Databases technology applies both OODB concepts and relational database, where data is stored in tables with labeled columns.

2.3.1 Normal Form

The process of normalization involves identifying relations to reduce redundancy and provide a more efficient database arrangement [4]. Table 3 shows some types of normal form relations, which are introduced by Codd [4]. First Normal Form (1NF) exist where no composite value is allowed, such as where DEP, DNum , Prof consist of Computer , 7, and {Tom, Kim} respectively in one row. Instead an additional primary key is created using Prof, therefore allowing the records to be in 1NF.

Table 3: Normal Forms

Form	Meaning	Example					
First (1NF)	Relation contains no non-atomic attributes or nested relations	DEP		DNum		Prof	
		Computer		7		Tom	
		Computer		7		Kim	
Second (2NF)	Relations where primary key contains multiple attributes	enum	tel	pay	tel	dname	loc
		1025	8568945	58	8568945	COMP	ND
		1056	7768945	75	7768945	MATH	ND
Third (3NF)	Relation should not transitively dependency on a non-key attribute on the primary key	ENUM	DOB	TEL	TEL	DNAME	DNUM
		1025	5/8/81	8568945	8568945	COMP	C170
		1056	8/12/74	7768945	7768945	MATH	M100

The concept of Second Normal Form (2NF) is viewed with full functional dependency. For full functional dependency to exist, an attribute such as pay from Figure 3 depends completely on the primary key (pk) from enum and bnum, which is used to

determine the employee pay⁵. To represent this relationship in 2NF the use of an attribute project is created with a new table where enum is the pk1, bnum is pk2 and pay. Figure 3 show attributes that are associated with parts of the primary key which are fully dependent. Example Pay is associated with ENUM and TEL, therefore forming a table separate from DNAME and LOC which is associated with TEL.

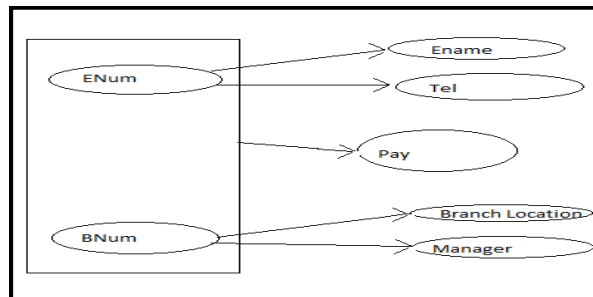


Figure 3: Functional Dependency

Third Normal Form (3NF) concept is based on transitive dependency. A 2NF relation can be in 3NF providing that attributes that are not a member of the primary key contains non-transitivity dependency on the primary key. The concept of transitive dependency is a functional dependency bound by transitivity. Three unique attributes A, B and C represent a relation.

$A \rightarrow B$ hold true

$B \rightarrow A$ hold false

$B \rightarrow C$ hold true

⁵ <http://www.emunix.emich.edu/~khailany/files/Normalization.htm>

The functional dependency $A \rightarrow C$ is based on 1 and 3 making it transitivity dependency.[17] The example provided in Table 3 looks at dependency ENUM to DNUM is transitive through TEL.

Boyce-Codd normal form (BCNF) is stricter than 3NF. All members of BCNF are members of 3NF however, not all 3NF is a member of BCNF. Members in BCNF are in 4NF. Consider a relation P exist:

$$a \rightarrow b \text{ and } a \rightarrow c$$

If b and c dependent on each other, P is in the form of BCNF⁶.

2.3.2 SQL

Structured Query Language (SQL) or Structured English Query Language (SEQUEL) is the interface for Relational Database designed at IBM Research⁷. With the introduction of SQL, it was a major contribution to the success of relational database, which emerged after network or hierarchical database system. It is the standard language for American National Standards Institute (ANSI) and International Standards Organization (ISO) [4]. Versions of SQL includes: SQL-86 or SQL1, SQL-92 or SQL2, SQL -99 or SQL3, SQL -03, SQL-06 and SQL-08 [5]. SQL: 2003 and SQL:2006 release feature XML component to SQL, while SQL 2008 allows order outside cursor definition.

⁶ <http://www.emunix.emich.edu/~khailany/files/Normalization.htm>

⁷ <http://www.almaden.ibm.com/cs/people/chamberlin/SQL-encyclopedia-entry.pdf>

Significant features were added to SQL-99, such as triggers, recursive queries, some object-oriented features and non-scalar types [6,4].

Other form of relational query languages includes, 4D Query Language, Datalog, HTSQL, QBE and Java Persistence Query Language (JPQL). SQL is used to manage, update and retrieve data making it a data definition language (DDL) and also a data manipulation language (DML). Relational model terms such as relational, tuple and attributes are replaced with the SQL terms such as table, row and column respectively.[4]. For this research SQL: 2003 will be query language used.

2.4 Model Transformation

According to M. Lamari [11] model transformation is viewed as taking a model as input applying some rules to it, which will produce an output that is a different representation of the input. Algorithms or rules are normally used to specify the model transformation process. Applying these rules and algorithms at times does not necessarily give the desired outcome, based on prior model being transformed.

With the ability to transform from one model to another would aid developers in getting a domain in the format that is required by a system. Applying transformation to a domain can transform a system from a low level e.g. a class diagram which is not understood by a system user to a database structure, which can be used by an average system user.

2.5 Related Work

In this section of the thesis, four related work will be presented which is of great significance to this research. Marcos, E., et al [14] work on how to extend UML for object relational database will be the first presented work, outlining the informal method. The second related work is that of Mok and Paper [15], who looks at how to transform form UML model to object relational model using a mathematical approach. This is essentially the formal methodology. The third work will be that of the work of R. Chennamaneni [7], which applied the formal and the informal methodologies to a particular domain. The final work is a description of the problem domain used in this work, i.e. Kalman Filter System.

2.5.1 Related Work 1

The work carried out by Marcos, E., et al. which looks at “Extending UML for Object-Relational Database Design” [14]. In database E/R model is often used to model a particular view of a system. Object-oriented design language UML allows the model of the entire system, making use of the extensible aspect of new stereotypes. The focus of Marcos et al is to establish new UML stereotypes for Object-Relational Database design. From this process a set of guidelines are used to translate UML schema into object-relational, using the SQL: 1999 object-relational as the blue print and Oracle 8i as the implementation example.

The methodology proposed consists of three phases: analysis, design and implementation. In the analysis phase, UML class diagrams are used to design conceptual schema. Within the design phase, it consist of two steps, standard and specific design. Standard design focus on a logical design, it can utilize two techniques, define the schema in SQL: 1999 and or use a graphical notation which is UML extensions. In the specific design the schema should be specify based on the SQL of the chosen product, in this case Oracle 8i is used. Within the implementation phase the actual design is implemented.

The researchers have tested this methodology with a local domain at their institution which deals with reserving computer in an university. The three phases were used to develop an application. In this research this methodology will be applied to the domain being used. The rest of the work will refer to this work as informal method or methodology1.

2.5.2 Related Work 2

The work of Mok and Paper provides another methodology which looks at,“ On Transformations from UML Models to Object-Relational Databases”[15]. The work covered looked at both static aspects as well as the dynamic aspect of modeling. At this stage of the research we are the focusing on the static aspect. Within the static aspect the process involves the removal of semantically overloaded elements, and then applying an algorithm to generated nested normal form nested tables.

The process of UML to Object-Relational Databases transformation focuses on the design view of the system which is captured in the class diagrams. For class diagrams to derive NNF nested tables, semantically overloaded UML elements should be removed. Some assumptions are made about a class, such as a class in a class diagram plays one role, UML does not accept nulls. Another assumption is a class and its relationship is in BCNF.

The next stage, of this transformation focused on the use an algorithm, (Algorithm 0) to account for relationship in BCNF. The result from Algorithm 0 is a semantically equivalent which converts role names and association to subclasses which is used as the input to another algorithm (Algorithm 1). Algorithm 1 focuses on association as the only structural relationship and removing the other relationship which are not needed for data storage. The results from this algorithm are nested tables with attributes added. SQL:1999 statement is then developed from these nested tables.

The rest of the work will refer to this work as formal method or methodology2. This methodology will be applied to the current domain.

2.5.3 Related Work 3

The thesis work done by R. Chennamaneni [7] form the bases on which this research will be continued. The use of an airline reservation system was used in the model transformation of class diagrams to object relational schema in SQL:2003 using both an informal and formal method. Informal method focused on the use of UML modeling

concepts while the formal method focus on sound mathematical principles such as graph theory.

Within the informal technique the transformation from UML diagram to object-relational schema. This was carried out by using UML extension mechanisms, to extend the class diagrams which are then represented in SQL statements. The formal technique removes semantically overloaded elements, by using an algorithm. Another algorithm uses this semantic equivalent class to produce nested relational schemas.

A comparison of the two methodologies was carried out, in a quantitative method. Quantitative comparison of the number of tables, reference type attributes, number of base level and others were done. The methodologies showed variants in most of the areas of comparison while, one area being compared was similar.

From the work done it was discovered that the informal method required fewer steps than the formal method. Based on examining the two methods the researcher used, the formal method required several additional steps as compared to the informal method for the transformation to be achieved. Another conclusion researched by the previous researcher outlined that formal method was more rigorous than that of the informal method. This was due to the mathematical nature of the transformation method. Formal method produces fewer tables as seen in Table 4, which contains complex structure while informal method produces more tables with simple structure.

The format of a class diagram is retained by the informal method, which aids in reverse engineering. The results of Table 4 produced a large number of tables from methodology 1, which will cause a high overhead for maintaining data integrity with the informal method while a low overhead is required with a formal method.

The work done was carried on one application domain, thus future work was recommended to be carried out on other domains to validate or contradict the result finding. Therefore, this researcher will be carrying out future validation of this work done in a different application domain, by applying the similar approach to the current problem domain.

Table 4: Quantitative comparison of AFR

	Methodology1	Methodology1
Number of base level types	16	16
Number of intermediate types	0	5
Number of tables	14	4
Number of reference type attributes	25	0
Oracle implementation specific types	7	5
Oracle implementation specific storage nested tables	13	4

2.5.4 Related Work 4

Kalman filter is essential to the estimation of the state of an object such as spacecraft, geographical system, or even how an aircraft. The nature of Kalman filter is established on a recursive algorithm that is constantly updated [13]. Gyroscopes is used to measure altitude but it degrades over time, so Kalman Filter provides optimal estimates which assist object staying on the desired course. This recursive algorithm provides estimation of a state such as its position, altitude or even noise characteristics.

The Kalman filter then handles the measurement aspect. Due to inability to carry out physical measurement, a relationship between the measurement model and the state is

carried out. Handling another limitation of this system which looks at measurement only available at a particular time a projection is made until new measurements are available.

An example was noted [13], about aircraft altitude estimation problem. With gyroscope accuracy decreasing over time a star-tracker is used to correct this. The Kalman filter would model such system where the gyroscope degrades and another aspect of the equation models the relationship between the star-tracker measurements. The Kalman filter would then estimate where there would be an inaccuracy in the gyroscope.

An UML class diagram was developed for the Kalman Filtering domain, which shows the static concepts which entails process model and measurement model. Process model consist of aggregations such as state, process or control while measurement models can be either linear or nonlinear. For this problem domain a class diagram is shown in Figure 4 of a Kalman Filter diagram [13].

CHAPTER 3

TRANSFORMATION METHODOLOGY

3.1 Description of Methodologies

3.1.1 Methodology I

Methodology I applies the concepts of Informal Transformation Technique (ITT), to transform UML class diagram to object relational database concepts of SQL:2003, utilizing UML extension mechanisms such as stereotypes, constrains and tagged values.

Table 5: Stereotypes for database design [12]

View	Database Element	UML Element	Stereotype
Architectural	Database	Component	<<Database>>
	Schema	Package	<<Schema>>
Persistent class	Persistent class	Class	<<Persistent>>
	Multivalued Attribute	Attribute	<<MA>>
	Calculated Attribute	Attribute	<<DA>>
	Composed Attribute	Attribute	<<CA>>
	Identifier	Attribute	<<ID>>
Logical	Table	Class	<<Table>>
	View	Class	<<View>>
	Column	Attributes	<<Column>>
	Primary Key	Attribute	<<PK>>
	Foreign Key	Attribute	<<FK>>
	NOT NULL Constraint	Attribute	<<NOT NULL>>
	Unique Constraint	Attribute	<<Unique>>
	Trigger	Constraint	<<Trigger>>
	CHECK Constraint	Constraint	<<Check>>
	Stored Procedure	Class	<<Stored Procedure>>
Physical	Table space	Component	<<Tablespace>>
	Index	Class	<<Index>>

Presented in Table 5, are the relational model general representation of primary key, foreign keys, etc. [12]

Table 6: UML extension

<p>Description To model domains with object-relational databases, UML defines a set of tagged values, stereotypes and constrains. These extensions are specific to object-relational models which allow us represent them with existing system diagrams. These extensions are based on SQL: 1999 and Oracle 8i object-relational models. UML representation is corresponded to each element in the object-relational model. Stereotypes element are selected based on certain criteria. SQL: 1999 the focus on structured types and typed tables, which are defined in SQL schema. Oracle 8i considers object types, object tables and nested tables as SQL: 1999.</p>	
<p>Extensions Prerequisites We consider that the required extension for the relational mode has being defined in Table 3.1</p>	
<p>SQL: 1999 Stereotypes</p>	
<p>Structured Type Metamodel class: Class <i>Description:</i> A <<UDT>> allows the representation of new user defined Data Types <i>Constraints:</i> Maybe used to define value types. Tagged values: None</p>	<p>Typed Table Metamodel class: Class. <i>Description:</i> Defines as <<Object Type>>. It models database schema. <i>Constraints:</i> A typed table implies the definition of a structured type, which is the type of the table. Tagged values: None</p>
<p>Knows <i>Description:</i> A <<Knows>> association is a special relationship that joins a class with a user defined data type<<UDT>>that is used with the class. It is uni-directional relationship, where an arrow represents the direction of the association. <i>Constraints:</i> Can only be used to join a <<Object Type>> with an <<UDT>> class.</p>	<p>REF Type <i>Description:</i> A <<REF>> represents a link to some <<Object Type>> class. <i>Constraints:</i> <<REF>> attributes can only refer to a <<Object Type>> class. <i>Tagged values:</i> The <<Object Type>> class to which it refers.</p>
<p>ARRAY Metamodel class: Attributes. <i>Description:</i> An <<Array>> represents an indexed and bounded collection type. <i>Constraints:</i> A <<Array>> can be of any data type except the <<Array>> type. <i>Tagged values:</i> The basic types of array. The number of elements.</p>	<p>ROW Type Metamodel class: Attributes. <i>Description:</i> A <<row>> type represents a composed attribute with a fixed number of elements; each of them can be of different data type. <i>Constraints:</i> Has no methods <i>Tagged values:</i> Element name and its data type.</p>
<p>Redefined Method Metamodel class: Method. <i>Description:</i> A <<redef>> method is an inherited method that is implemented again by the child class. Constraints: None. <i>Tagged values:</i> The list of parameters of the method with their data types. The data type returned by the method.</p>	<p>Deferred Method Metamodel class: Method. <i>Description:</i> A <<def>> method is a method that defers its implementation to its child classes. <i>Constraints:</i> It has to be defined in a class with children <i>Tagged values:</i> The list of parameters of the method with their data types. The data type</p>

Stereotypes for database design proposed for relational database design in Table 5

is lacking the ability to model object model with REF types or methods. To model objects

in database E. Marcos, et. al [12], proposed an UML extension in Table 6 for object-relational database design which facilitates the limitation of Table 5.

With the introduction of stereotypes, tagged values and constraints designers are able to model object-relational databases. This extension focus around SQL: 1999, where each element is associated with a UML representation. The general approach entails three main steps for object-oriented database design, which includes analysis, design and Implementation. At the analysis phase UML class diagrams are used in place of Extended E/R diagrams to design conceptual schema.

At the design phase some features from the analysis phase overlaps. Two approaches may be taken in the design phase, a standard design which is not associated with any product, or a specific design which gears towards a product, e.g. Oracle 11g which will be an aim of this research. The final phase is that of implementation which focuses on the actual physical design. This phase is overall with the analysis phase however refining of pervious stage is needed to improve the quality of the application.

With the aid of Table 7 which presents the guidelines for transforming from one phase to another in object-relational database design. [14] Figure 4 represents the UML class diagram for a Kalman Filter program which is in the conceptual design phase.

Figure 5 represents the logical design with UML extensions for SQL: 2003.

Table 7: Guidelines for object-relational database design. [14]

UML	SQL:1999
Class Class Extension	Structured Type Typed Table

Attribute Multivalued Composed Calculated	Attribute ARRAY ROW / Structured Type in Column Trigger/ Method
Association One- To -One One-To- Many Many-To -Many	REF/REF REF/ARRAY ARRAY/ARRAY
Aggregation	ARRAY
Generalization	Types/ Typed Tables

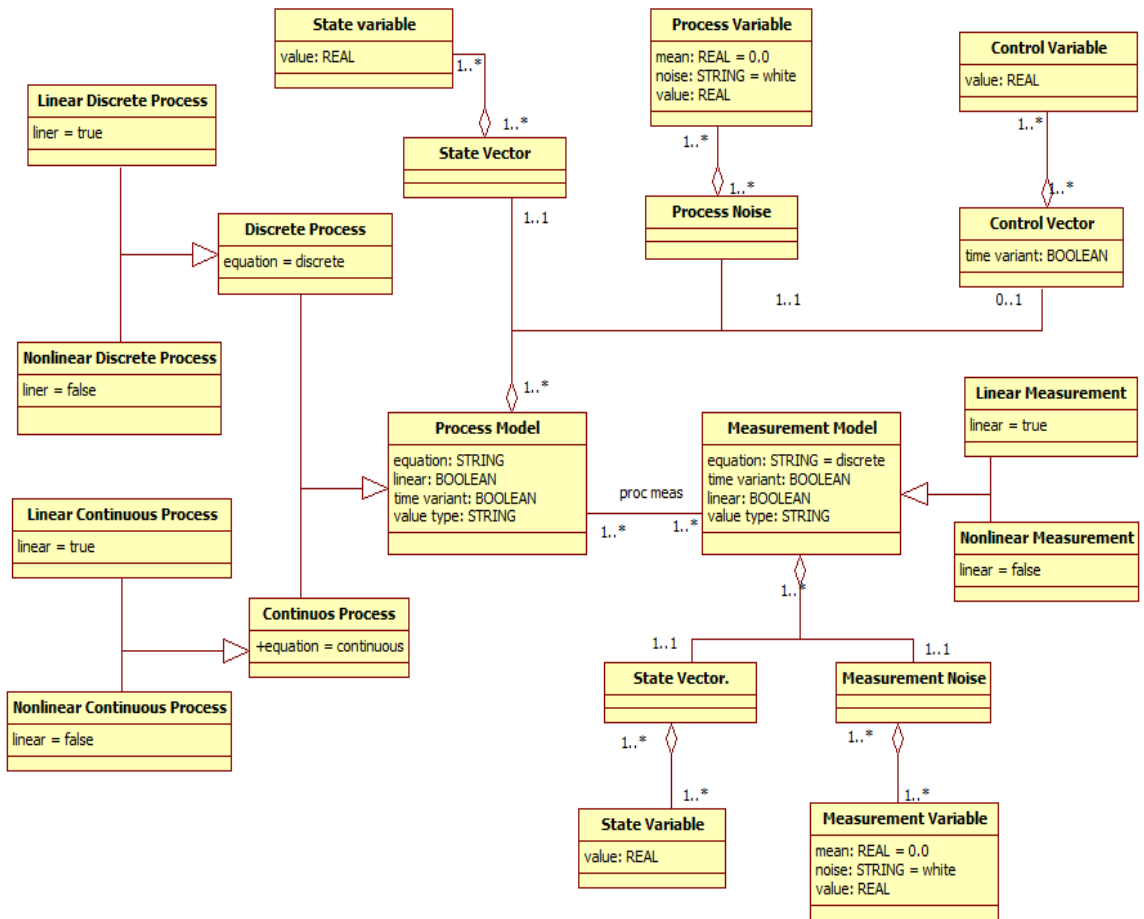


Figure 4: Class diagram of Kalman Filter Program

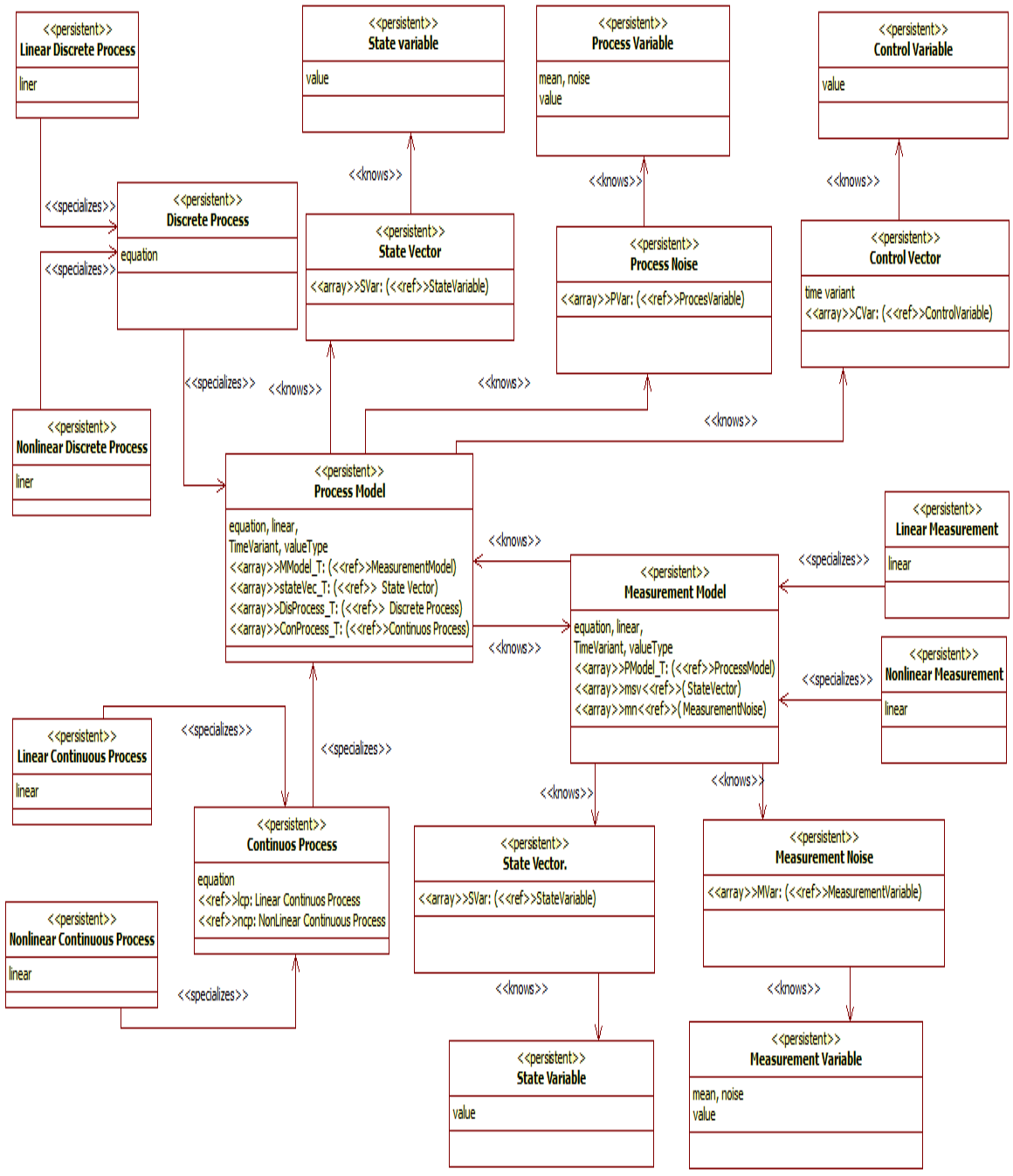


Figure 5: Extended Class Diagram of Kalman Filter Program

3.1.2 Methodology2

With the removal of semantically overloaded elements, this methodology applies a formal approach to transform UML class diagrams into SQL: 2003 statement. With the relationship criteria of BCNF met and having unique role for each class being satisfied, then the transformation can be carried out.

A first algorithm is used to remove semantically overloaded UML elements. The first stage of the algorithm creates sub-classes from associations and roles. The second stage is to examine each cycle, if a cycle hinders you from going around then it would pass this stage. However, if cycles exist then we add roles to the relationship until the Graham Reduction can succeed, where you are not able to navigate in a cycle.

Using the output from algorithm 1 as the input to algorithm 2, where we start with a node with no edge then remove one node at a time and add an edge between a corresponding graph. This will result in more than one possible reduction sequence. Another step in this algorithm is applied where an unmarked node exist until each is resolved. From this the nested relation schemas which are in NNF are generated.

3.2 Case Study Class Diagram

Kalman Filter program (KFP) class diagram which is obtain from [13] will be used to apply informal and the formal methodologies. The KFP consist of classes, attributes, association, multiplicities, specialization, generalization and aggregation. The KFP consist of a process model which shows multiple aggregations between state vector

process noise and control vector. Each aggregation has an individual multiple aggregation state variable, process variable and control variable respectively. Process model is also associated with one or multiple measurement model (vice versa) which consist of two aggregation state vector and measurement noise which are aggregated with state variable and measurement variable. Both process model and measurement model consist of generalization between subclasses.

Table 8: Stereotyped class of KFP

Class	Stereotype	Rationale
LinearDiscreteProcess	<<Object Type>>	Represented as a typed table in the database.
NonLinearDiscreteProcess	<<Object Type>>	Represented as a typed table in the database.
DiscreteProcess	<<udt>>	Represented as a user defined data type in the database used to define Linear Discrete Process, NonLinear Discrete Process.
StateVariable	<<Object Type>>	Represented as a typed table in the database.
StateVector	<<Object Type>>	Represented as a typed table in the database.
ProcessVariable	<<Object Type>>	Represented as a typed table in the database.
ProcessNoise	<<Object Type>>	Represented as a typed table in the database.
ControlVariable	<<Object Type>>	Represented as a typed table in the database.
ControlVector	<<Object Type>>	Represented as a typed table in the database.
ProcessModel	<<udt>>	Represented as a user defined data type in the database used to define Discrete Process, Continuous Process
MeasurementModel	<<udt>>	Represented as a user defined data type in the database used to define Linear Measurement, NonLinear Measurement
LinearMeasurement	<<Object Type>>	Represented as a typed table in the database.
NonLinearMeasurement	<<Object Type>>	Represented as a typed table in the database.
LinearContinuousProcess	<<Object Type>>	Represented as a typed table in the database.
NonlinearContinuousProcess	<<Object Type>>	Represented as a typed table in the database.
ContinuousProcess	<<udt>>	Represented as a user defined data type in the database used to define Linear Continuous Process, NonLinear Continuous Process
StateVector	<<Object Type>>	Represented as a typed table in the database.
MeasurementNoise	<<Object Type>>	Represented as a typed table in the database.
StateVariable	<<Object Type>>	Represented as a typed table in the database.
MeasurementVariable	<<Object Type>>	Represented as a typed table in the database.

3.3 Applying Methodologies

3.3.1 Methodology 1

Implementing methodology 1 to KFP class diagram in Figure 4 produced the extended class diagram in Figure 5. Standard classes stereotype are referred as <<Object Type>> while classes that are user defined stereotype are <<udt>>, which are outlined for the KFP classes in Table 8. Table 9 further presents the attributes of the KFP extended class diagram in Figure 5.

Table 9: Stereotyped attributes of Kalman Filter Program

Attributes	Stereotype	Rationale
StateVector svar	<<array>>, <<ref>>	Array of links to State Variable object.
ProcessNoise pvar	<<array>>, <<ref>>	Array of links to Process Variable object.
CVector cvar	<<array>>, <<ref>>	Array of links to Control Variable object.
ProcessModel mmodel_t svector pnoise cvector	<<array>>, <<ref>> <<array>>, <<ref>> <<array>>, <<ref>> <<array>>, <<ref>>	Array of links to Measurement Model object. Array of links to StateVector object Array of links to ProcessNoise object Array of links to ControlVector object
MeasurementModel pmodel_t msvect mnoise	<<array>>, <<ref>> <<array>>, <<ref>> <<array>>, <<ref>>	Array of links to Process Model object. Array of links to State Vector object. Array of links to Measurement Noise object.
StateVector msvar	<<array>>, <<ref>>	Array of links to State Variable object.
MeasurementNoise mvar	<<array>>, <<ref>>	Array of links to Measurement Variable object.

In the Kalman Filter Program class diagram of Figure 4, there exist several generalization a sample is selected from the KFP shown in Figure 6a where the

generalization between linear Discrete Process and nonlinear discrete process is discrete process. To handle such generalization, a specialization is created to handle each generalization pointing to the generalization discrete process which is shown in Figure 6b. A simple process is followed for the rest of generalization relationship.

Aggregation is another type of classification that is found in the KFP. To handle the relationship between Process model and state vector in Figure 6c, a <<knows>> is being introduced in Figure 6d which is semi directional pointing from the diamond connection between the classes. A one-many relationship exist between state vector and process model in Figure 6c, the results from Table 9 stereotype, is applied to the diagram which extends it to for Figure 6d.

The association relationship was also found in KFP, which is shown in Figure 6e. Figure 6f shows directional <<knows>> being introduced to capture the relationship of an association. Both classes carries attributes of each other based on the results from Table 9 about process and measurement model stereotype.

Figure 6: Class diagram and extended class diagram of KFP

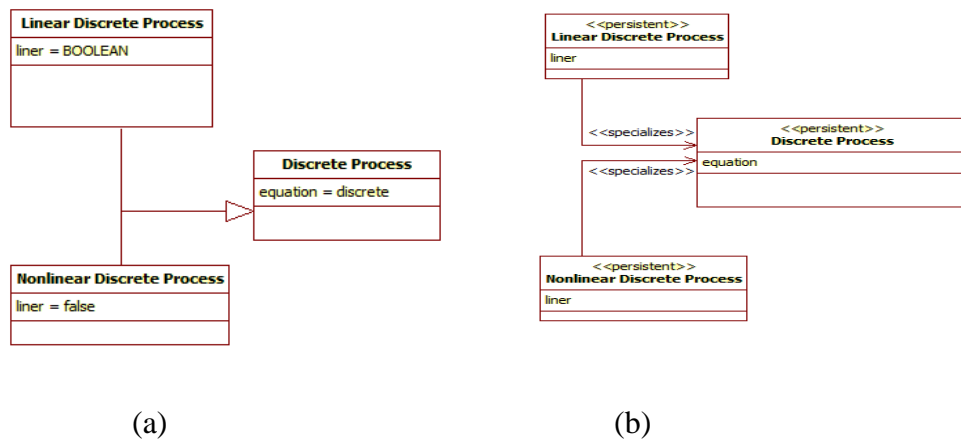
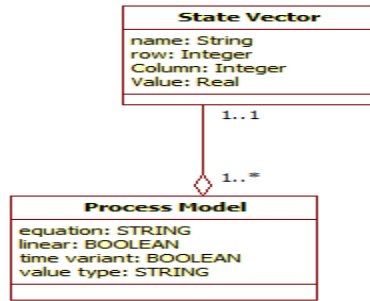
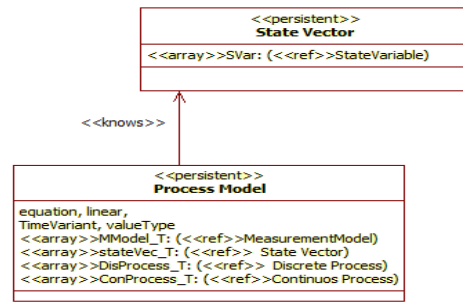


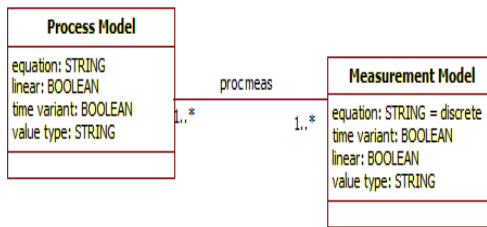
Figure 6: cont.



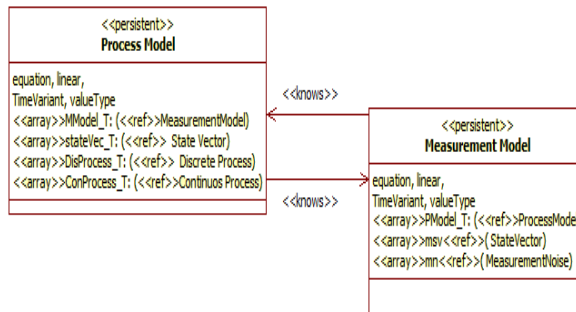
(c)



(d)



(e)



(f)

3.3.2 Methodology2

To apply Algorithm A on Figure 4 (KFP class diagram), identify semantically equivalent classes is needed. In this domain there is no semantically equivalent classes therefore the Graham Reduction processes is not required.

The class diagram in Figure 4 is used to produce nested tables by applying Algorithm B. The relations in Figure 4 are, $R = \{SvaSv, SvPm, PmPn, PnPv, PmCv, CvCva, PmM, MMsv, MsvMstv, MMn, MnMv\}$

Table 10: Abbreviation of stereotyped classes of KFP

Ld = LinearDiscreteProcess	Nd = NonLinearDiscreteProcess
Dp = DiscreteProcess	P = ProcessModel
Lp = LinearContinuousProcess	Np = NonLinearContinuousProcess
Cp = ContinuousProcess	Sv = StateVector
Sv = StateVariable	Pn = ProcessNoise
Pv = ProcessVariable	Cv = ControlVector
Cv = ControlVariable	M = MeasurementModel
Lm = LinearMeasurement	N = NonLinearMeasurement
S = MeasurementStateVector	Vr = MeasurementStateVariable
Mn = MeasurementNoise	Mv = MeasurementVariable

Figure 7: Hyper graph of relations in the case study

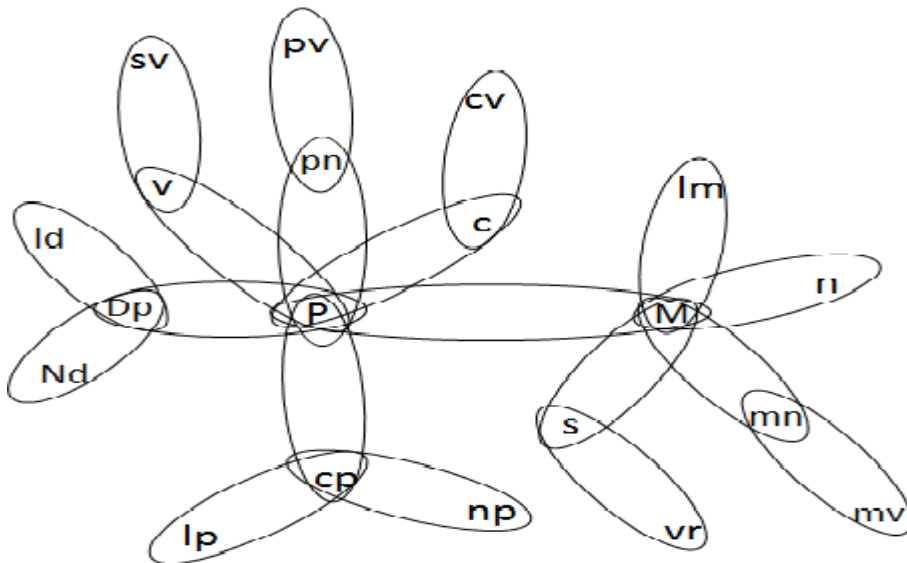


Fig 7 cont.

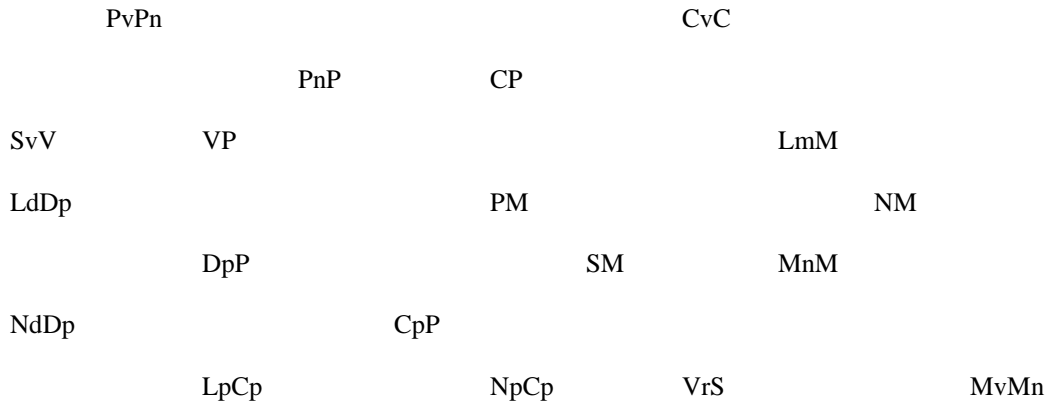


Figure 8: Graham Reduction process of KFP

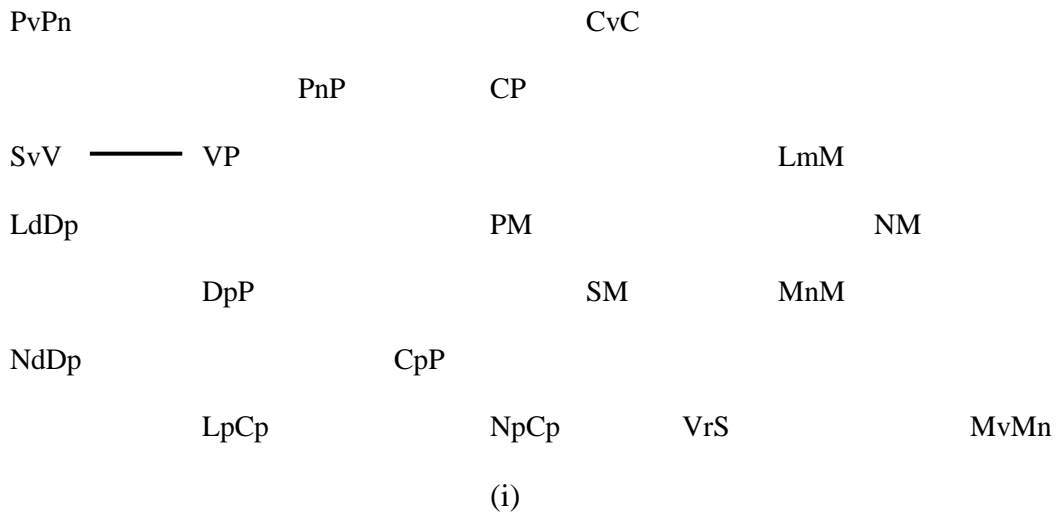
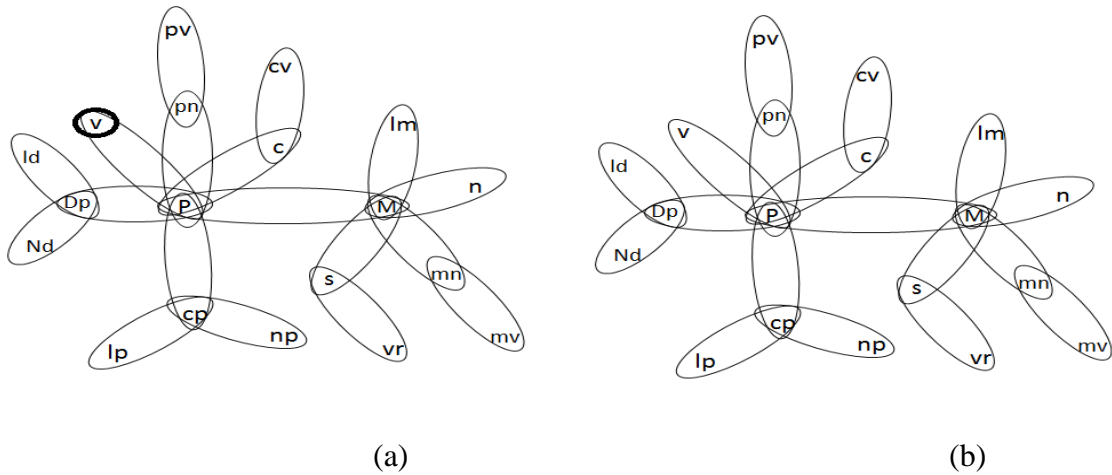


Figure 8: cont.

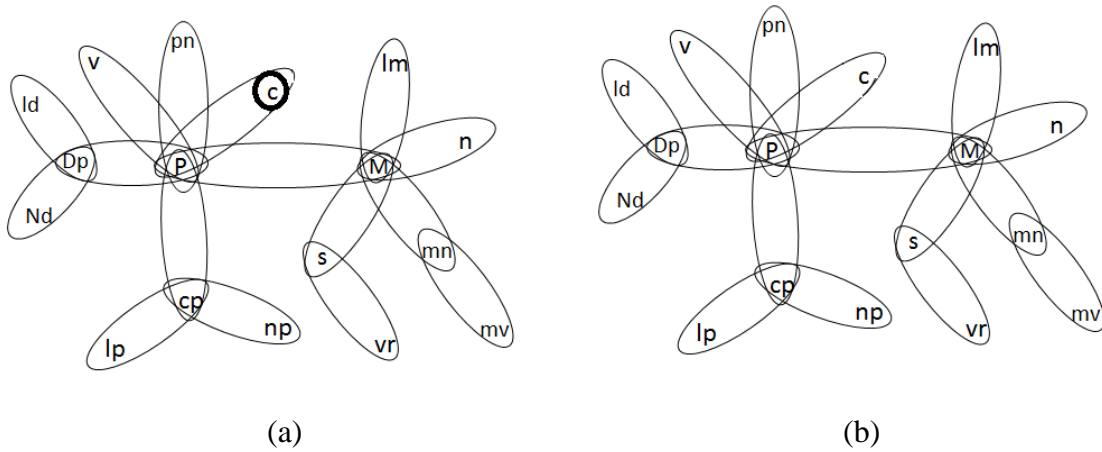
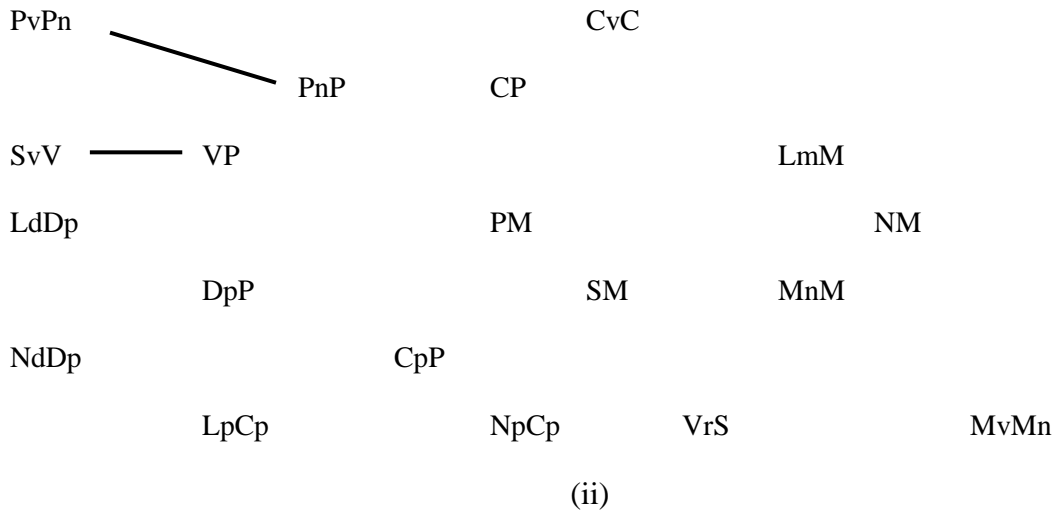
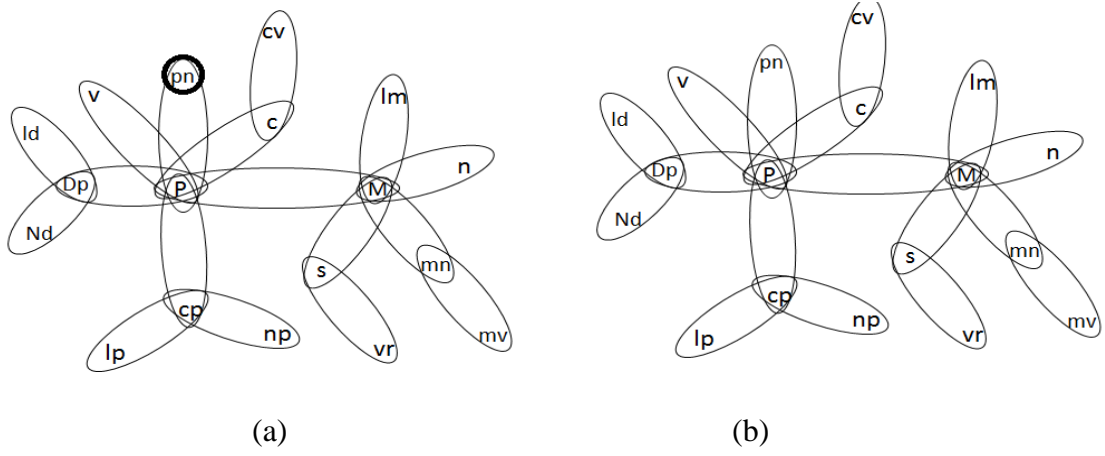


Figure 8: cont.

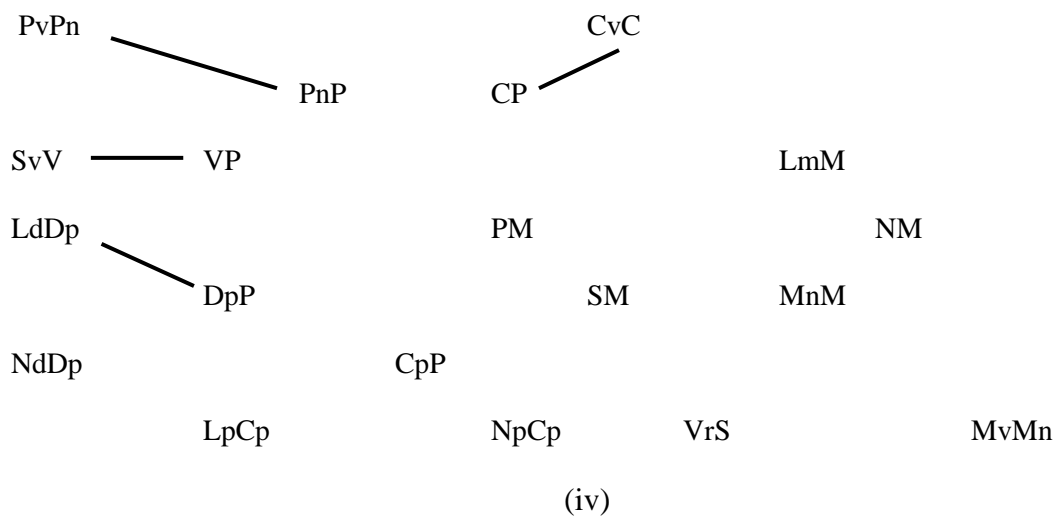
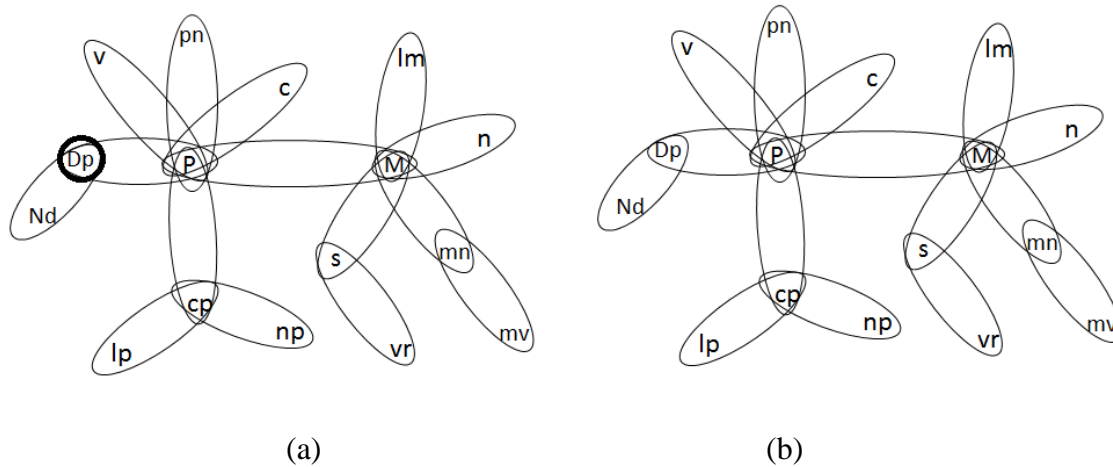
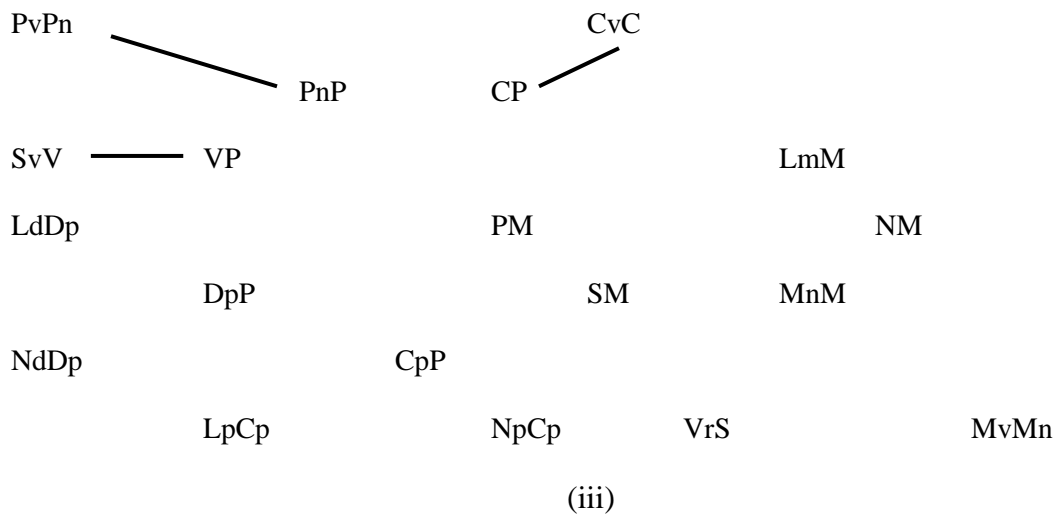
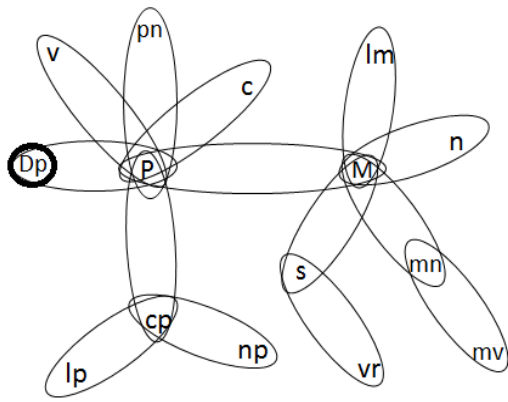
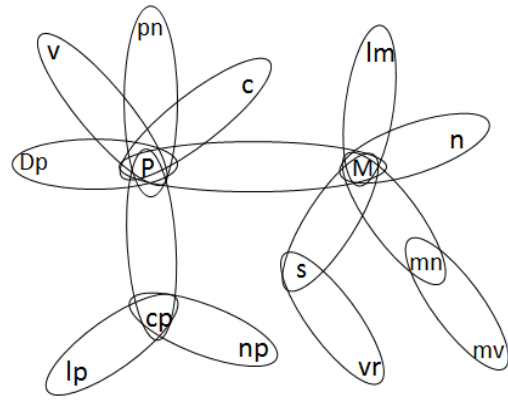


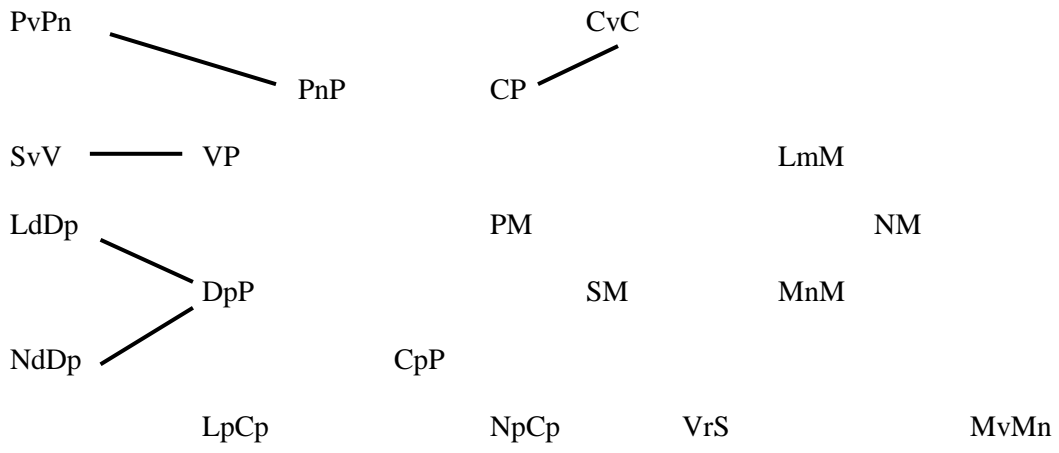
Figure 8: cont.



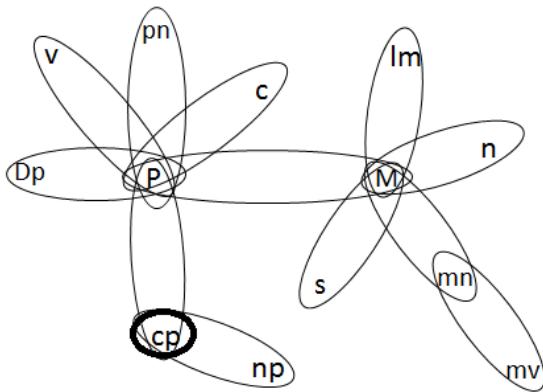
(a)



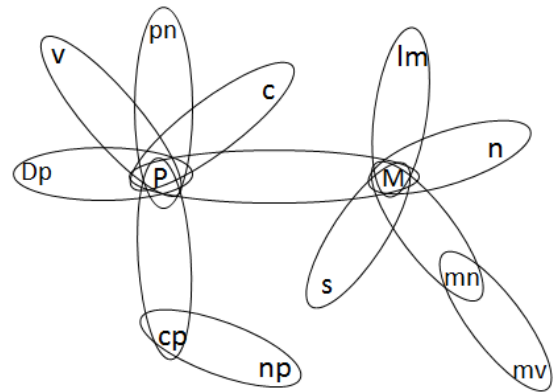
(b)



(v)



(a)



(b)

Figure 8: cont.

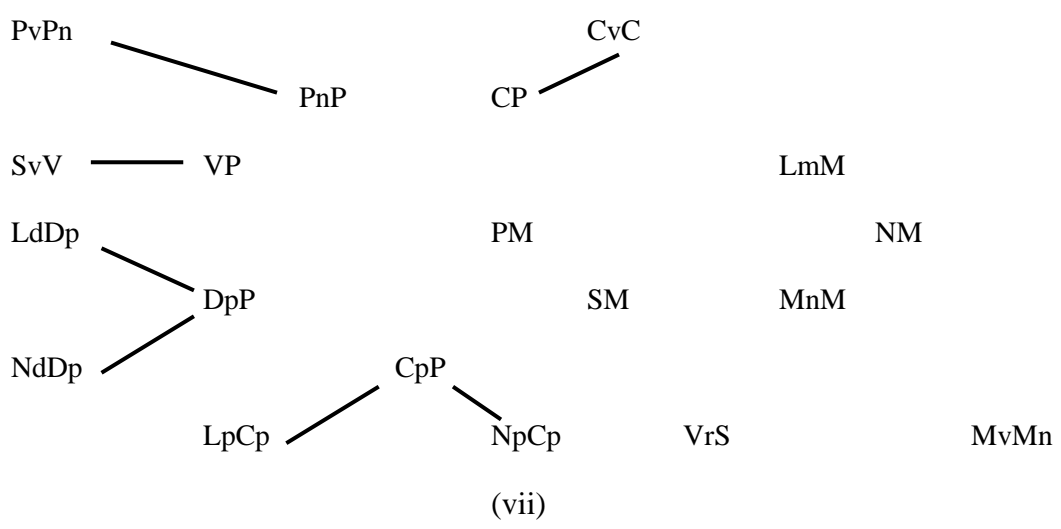
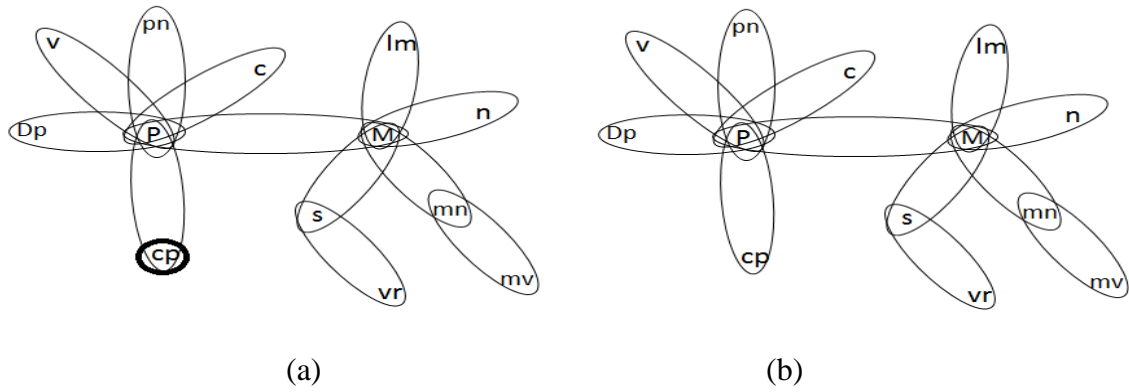
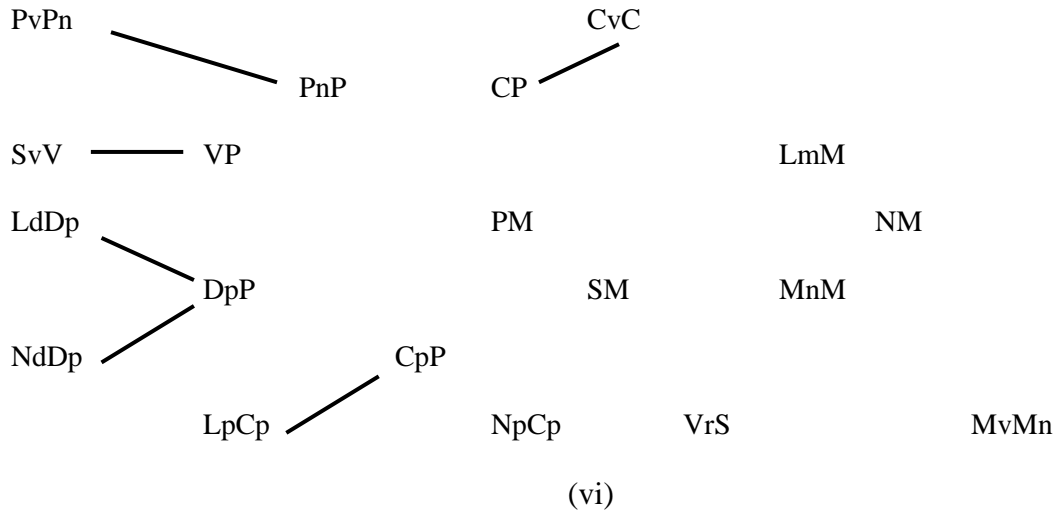
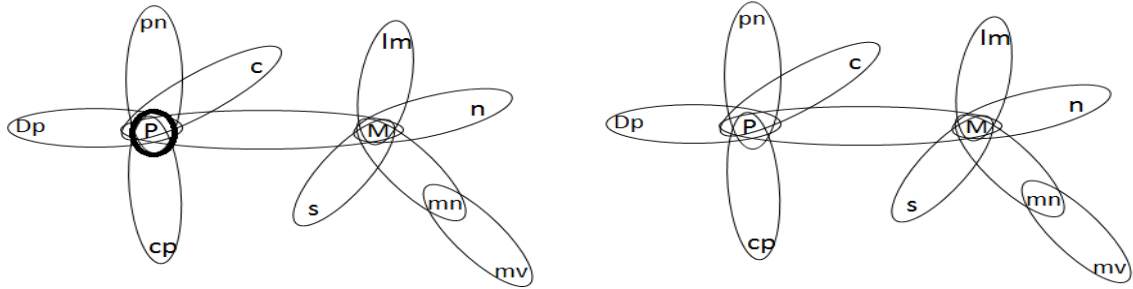
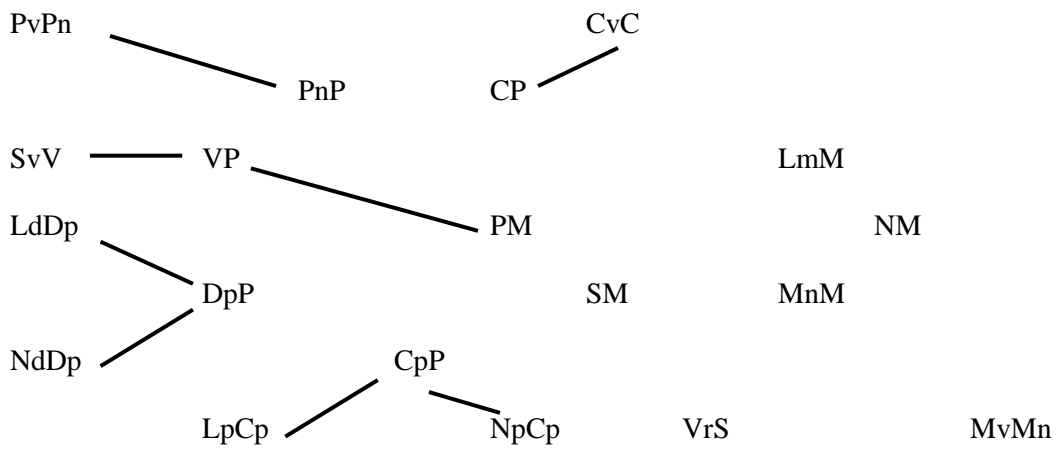


Figure 8: cont.

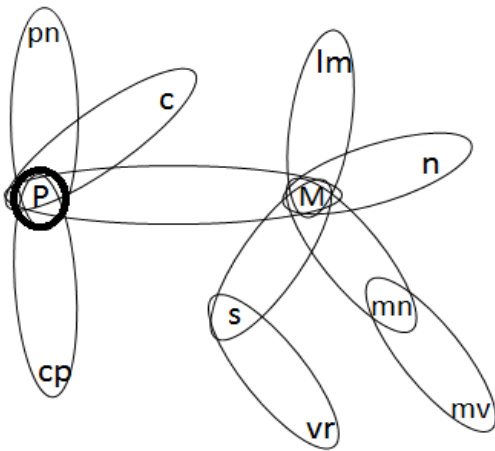


(a)

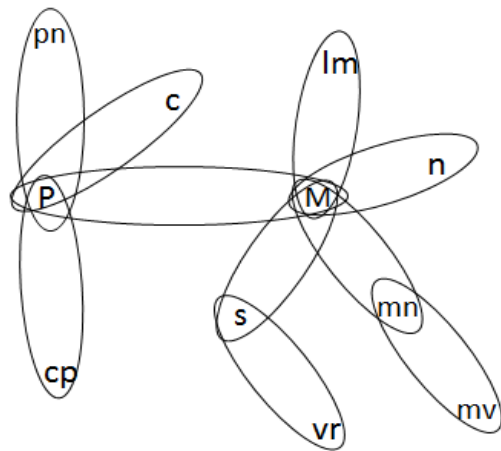
(b)



(viii)



(a)



(b)

Figure 8: cont.

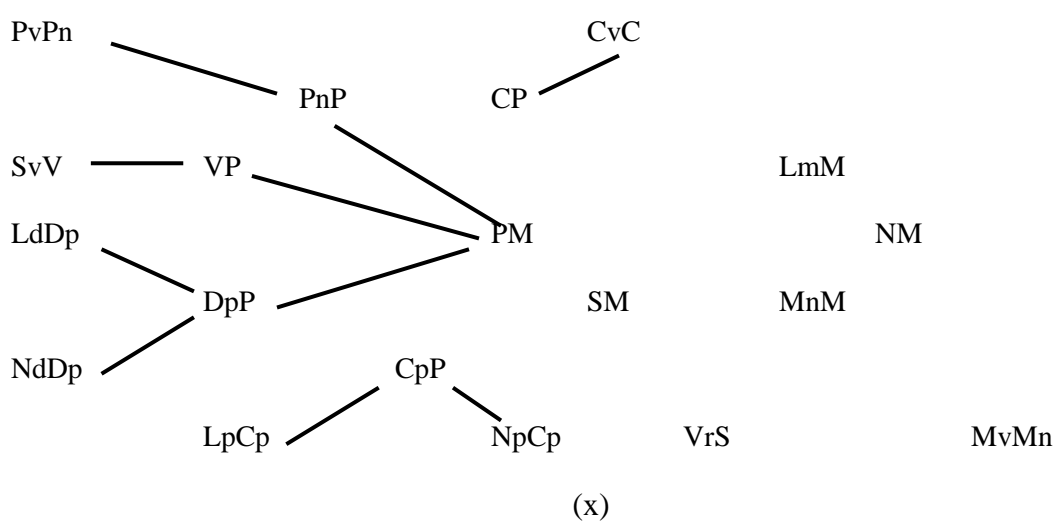
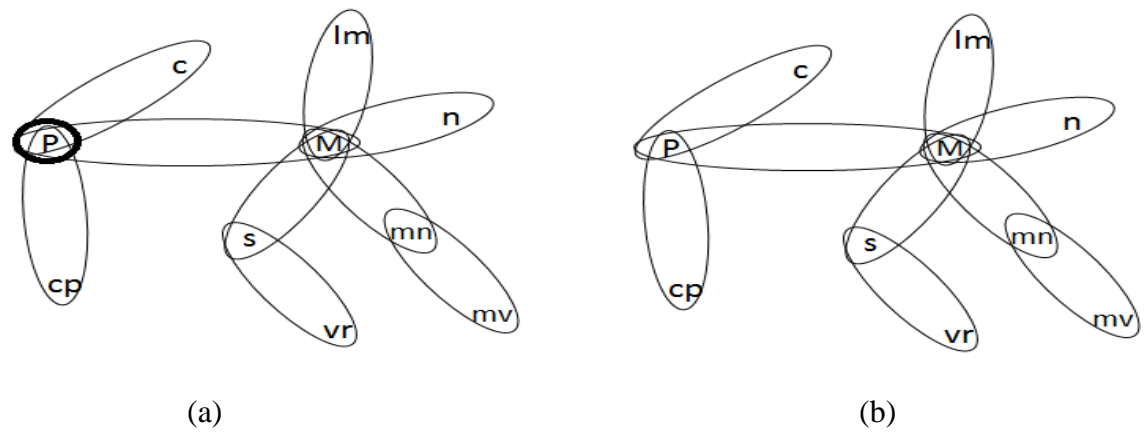
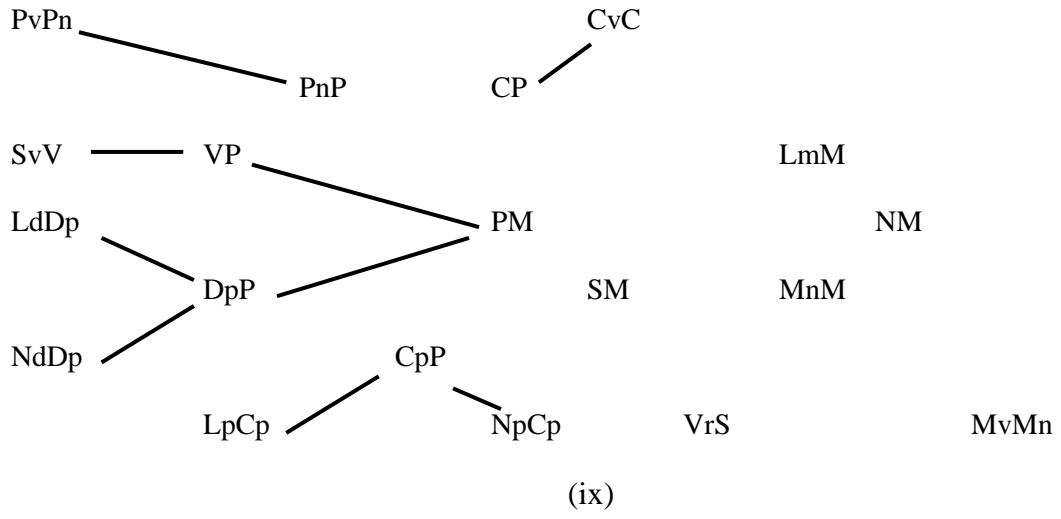
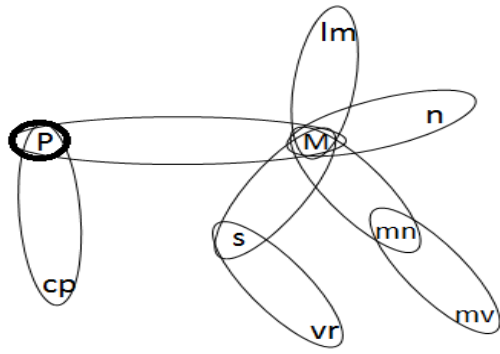
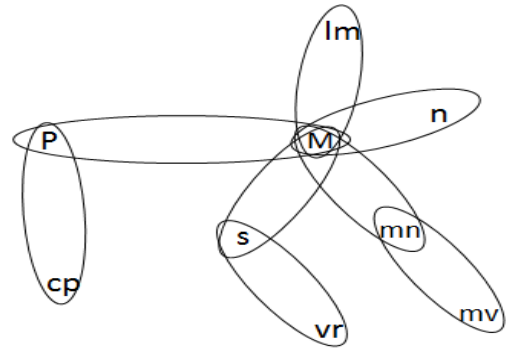


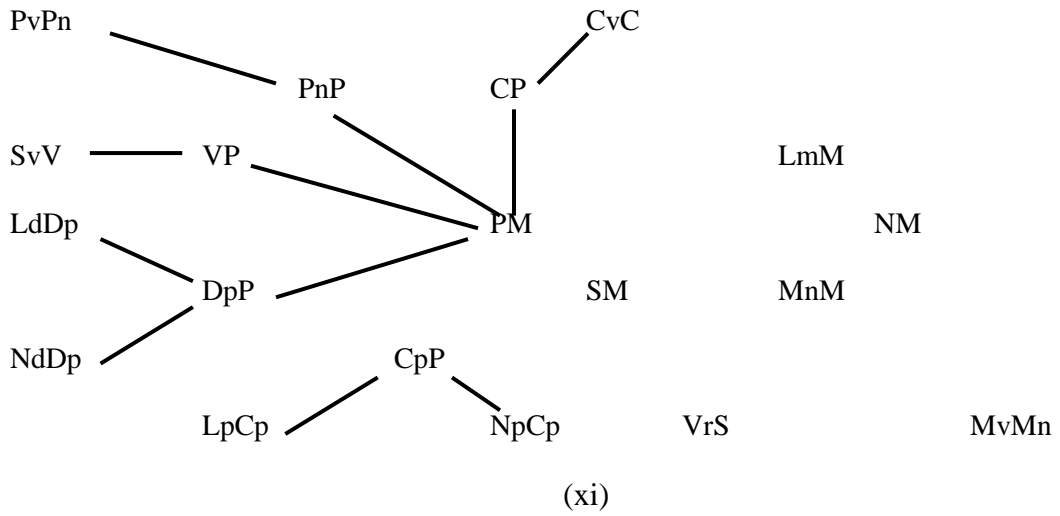
Figure 8: cont.



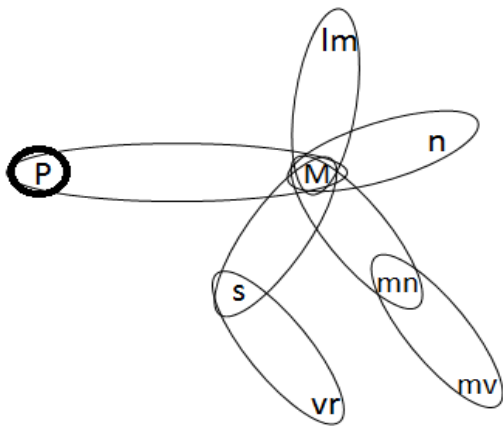
(a)



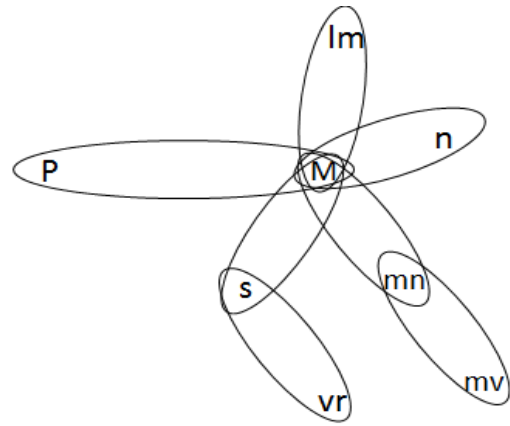
(b)



(xi)



(a)



(b)

Figure 8: cont.

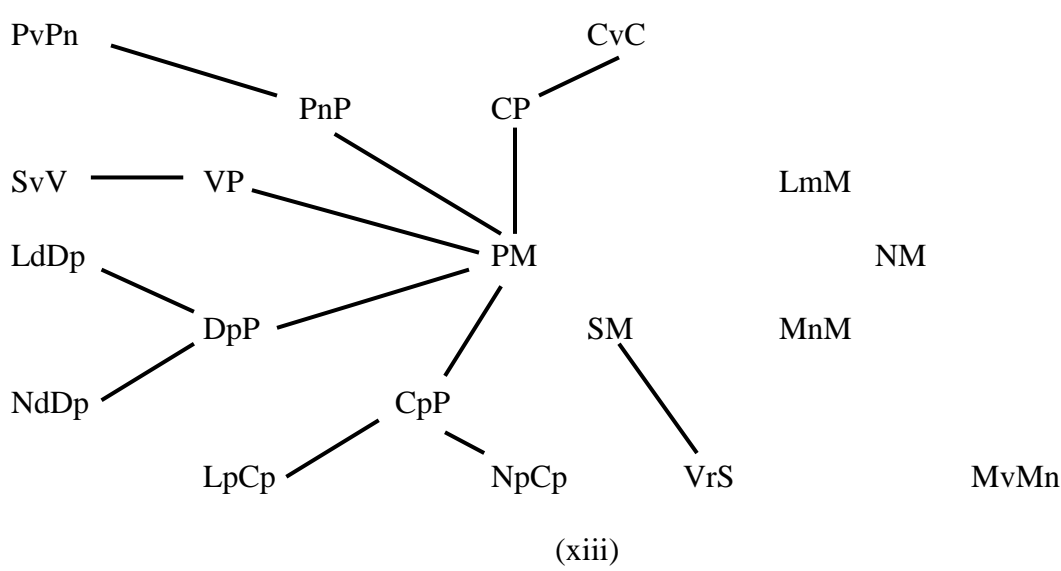
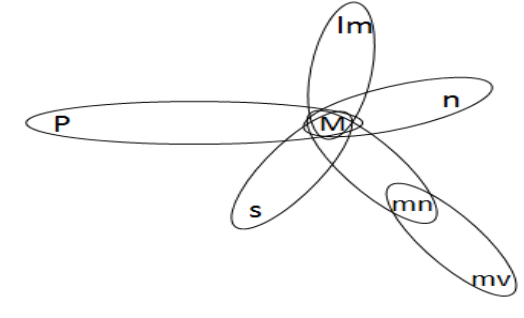
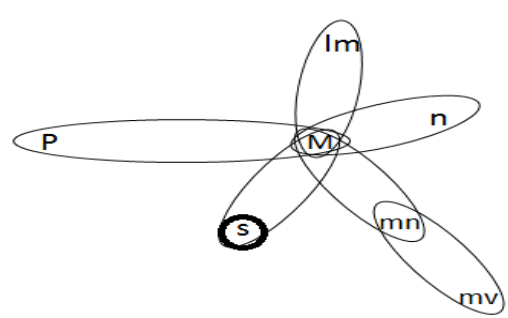
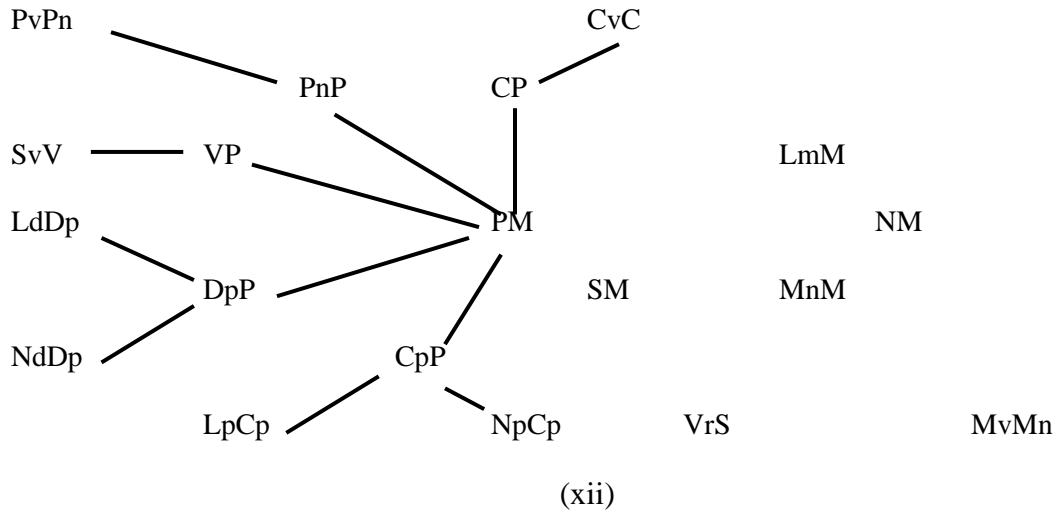
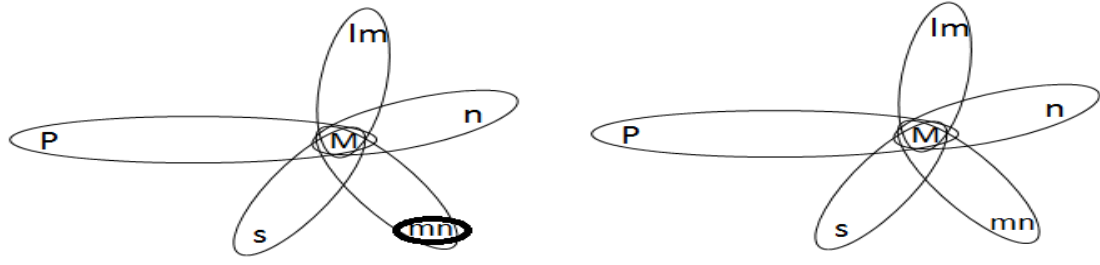
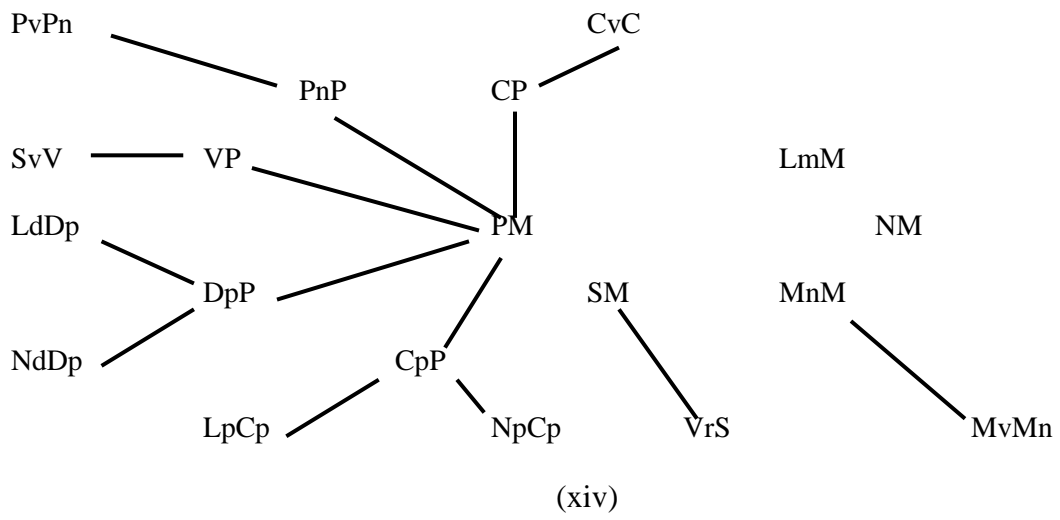


Figure 8: cont.

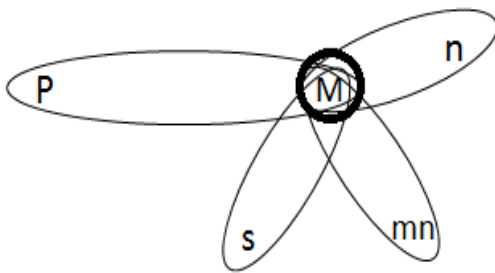


(a)

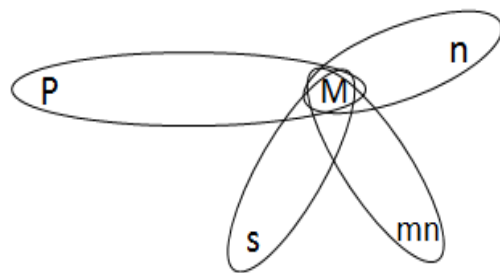
(b)



(xiv)

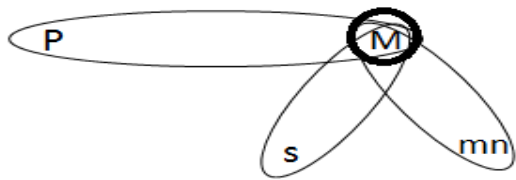
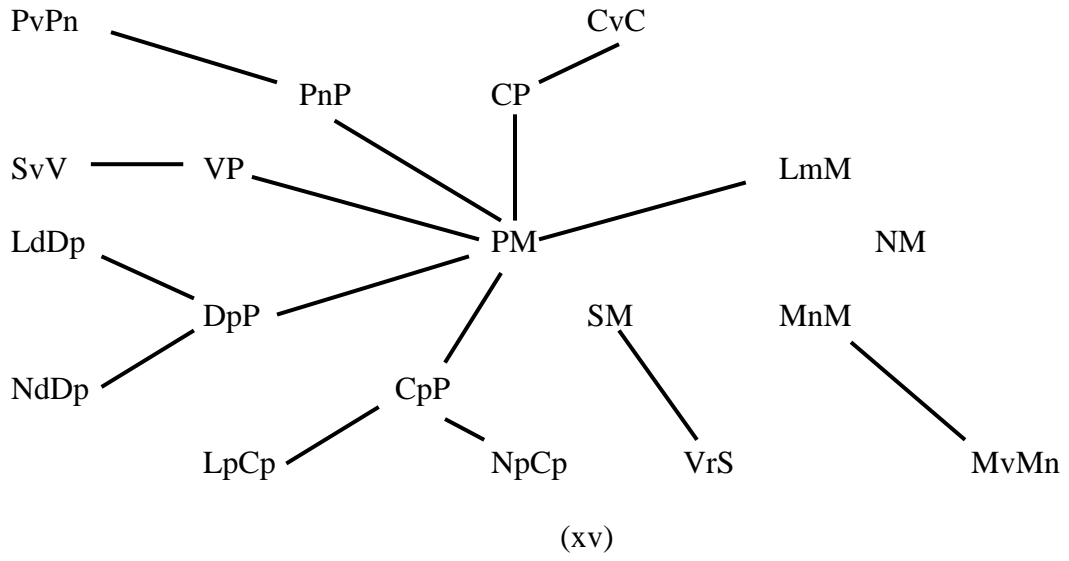


(a)

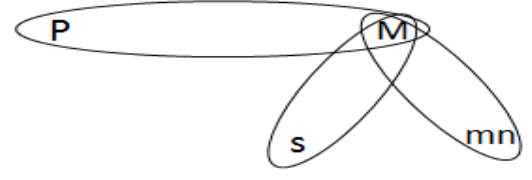


(b)

Figure 8: cont.



(a)



(b)

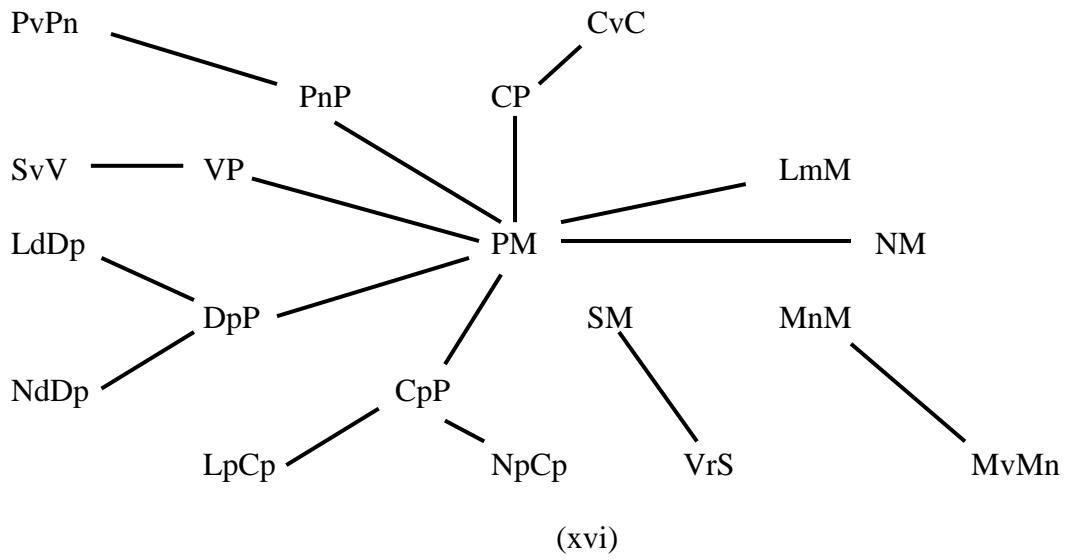
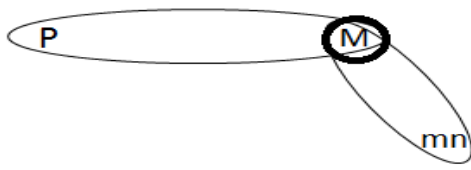
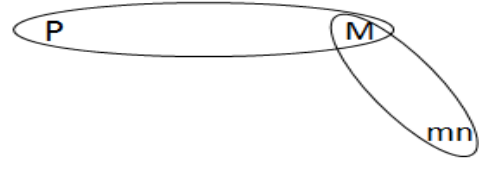


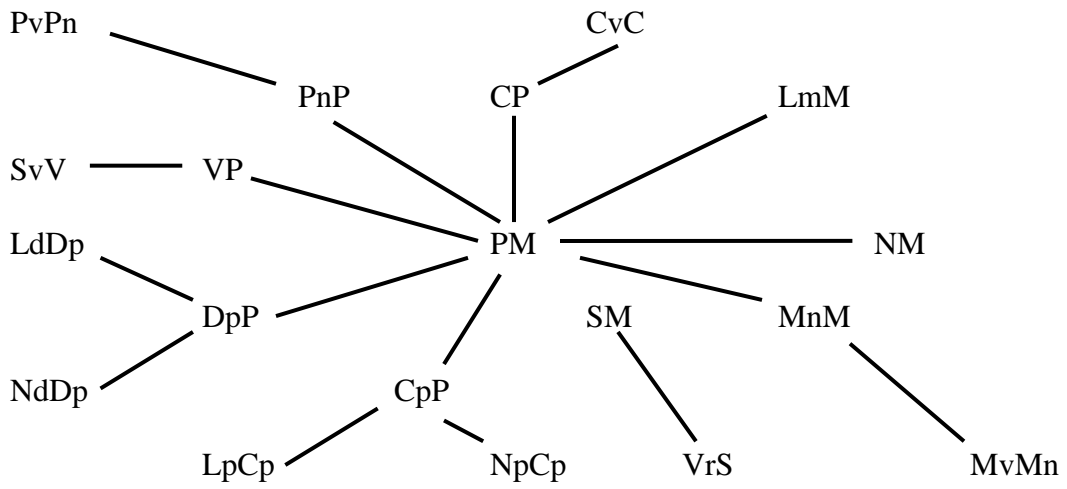
Figure 8: cont.



(a)



(b)



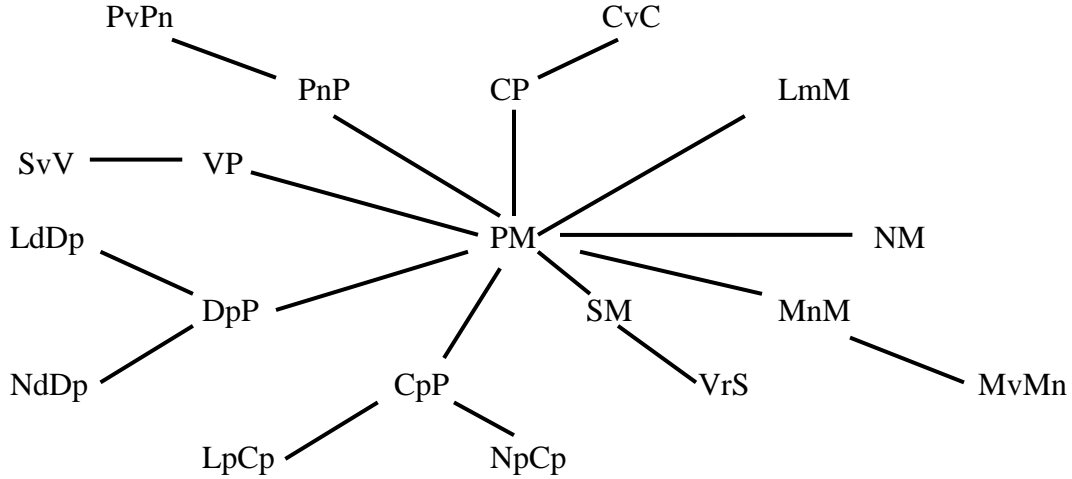


Figure 9: Join tree of KFP

The join tree in Figure 9 was derived from applying algorithm B step 1 to Figure 7 hyper graph and relation. The steps involves in building the join tree was adding an edge whenever the hyper graph was reduced. The hyper graph PvPn in Figure 8(ii) (a) edge is then removed to produce Figure 8(ii) (b) where it is part of PnPm hyper graph. As a result of this process, the join tree was able to add an edge between PvPn and PnPm in Figure 8. The rest of edges added to the join tree were derived from going through similar process with the hyper graph in Figure 7.

Step two of algorithm B produces database scheme as the output from using Figure 9 join tree as the input. Step 2.1 PvPn is used as the seed relation scheme R_{seed} , a single path scheme tree is created with the seed PvPn. The next stage is to mark the node PvPn and add it to L. At stage 2.2.1 the first marked relation scheme PvPn from L as R_M is removed. With PvPn containing one unmarked node R_U is PvPm and node N is PvPn it satisfies $R_U \rightarrow \text{Ancestor}(N)$ and $(R_U \cap R_M) \subseteq \text{Ancestor}(N)$, which $PvPm \rightarrow PvPn$ and $(PvPm \cap PvPm)$ is a subset of PvPm. Pm is attached as the child of N which is

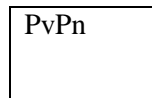
PvPn. PvPm is now entered into L. PvPm now becomes RM, where the unmarked neighbors are PmSv, PmCv PmM. These unmarked neighbors will require the repeat of process from step 2.1.

Table 11 shows the process for producing the nested table. The use of L, R_M, R_U Ancestor (N) at different stages of the Algorithm B. L is further broken into two parts L_s and L_e where L_s represent at the starting stage while L_e represents at the end of a stage.

Table 11: Application of Algorithm B

Iteration 1 of step 2:

(2.1) L_s = PvPn;

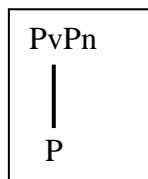


(2.2.1) L_s =; R_M = PvPn;

(2.2.2.1) L_s =; R_M = PvPn; R_U = PnP; N = PvPn; Ancestor (N) = PvPn

L_e = PnP

PnP → PvPn

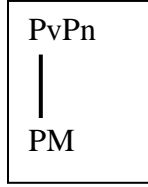


(2.2.1) L_s =; R_M = PnP;

(2.2.2.1) L_s =; R_M = PnP; R_U = PM; N = P; Ancestor (N) = PPvPn;

L_e = PM

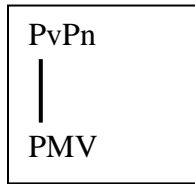
PM → PPvPn and PPvPn → PM



(2.2.2.1) $L_s = PM$; $R_M = PvP$; $R_U = PV$; $N = PM$; Ancestor (N) = PMPvPn;

$L_e = PM, PV$

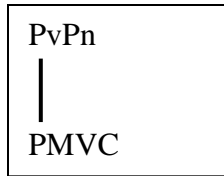
$PV \rightarrow PMPvPn$ and $PMPvPn \rightarrow PV$



(2.2.2.1) $L_s = PM, PV$; $R_M = PvP$; $R_U = CP$; $N = PMV$; Ancestor (N) = PMVPvPn;

$L_e = PM, PV, CP$

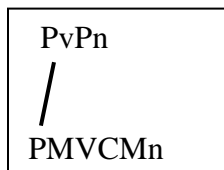
$CP \rightarrow PMVPvPn$ and $PMVPvPn \rightarrow CP$



(2.2.2.1) $L_s = PM, PV, PC$; $R_M = PvP$; $R_U = MMn$; $N = PMVC$;

Ancestor (N) = PMVCPvPn; $L_e = PM, PV, CP, MMn$

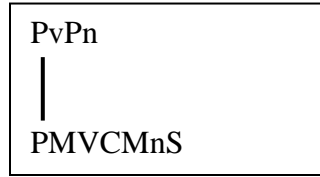
$MMn \rightarrow PMVCPvPn$ and $PMVCPvPn \rightarrow MMn$



(2.2.2.1) $L_s = PM, PV, PC, MMn$; $R_M = PvP$; $R_U = MS$; $N = PMVCMn$;

Ancestor (N) = PMVCPvPnMn; $L_e = PMn, PV, CP, MMn, MS$

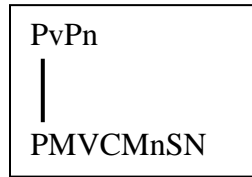
MS → PMVCPvPnMn and PMVCPvPnMn → MS



(2.2.2.1) $L_s = \text{PM, PV, PC, MMn, MS}$; $R_M = \text{PvP}$; $R_U = \text{NM}$; $N = \text{PMVCMnS}$;

Ancestor (N) = PMVCPvPnMnS; $L_e = \text{PMn, PV, CP, MMn, MS, NM}$

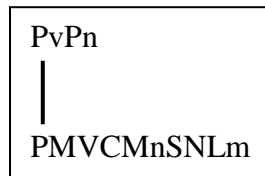
NM → PMVCPvPnMnS and PMVCPvPnMnS → NM



(2.2.2.1) $L_s = \text{PM, PV, PC, MMn, MS, NM}$; $R_M = \text{PvP}$; $R_U = \text{LmM}$; $N = \text{PMVCMnSN}$;

Ancestor (N) = PMVCPvPnMnSN; $L_e = \text{PMn, PV, CP, MMn, MS, NM, LmM}$

LmM → PMVCPvPnMnSN and PMVCPvPnMnSN → LmM



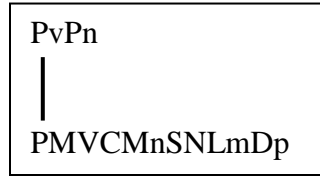
(2.2.2.1) $L_s = \text{PM, PV, PC, MMn, MS, NM, MLm}$; $R_M = \text{PvP}$; $R_U = \text{DpP}$; $N =$

PMVCMnSNLm;

Ancestor (N) = PMVCPvPnMnSNLm; $L_e = \text{PMn, PV, CP, MMn, MS, NM,}$

LmM, DpP

$DpP \rightarrow PMVCPvPnMnSNLm$ and $PMVCPvPnMnSNLm \rightarrow DpP$

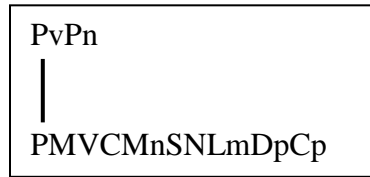


(2.2.2.1) $L_s = PM, PV, PC, MMn, MS, NM, MLm, DpP$; $R_M = PvP$; $R_U = CpP$;

$N = PMVCMnSNLmDp$; Ancestor (N) = $PMVCPvPnMnSNLmDp$;

$L_e = PMn, PV, CP, MMn, MS, NM, LmM, DpP, CpP$

$CpP \rightarrow PMVCPvPnMnSNLmDp$ and $PMVCPvPnMnSNLmDp \rightarrow CpP$

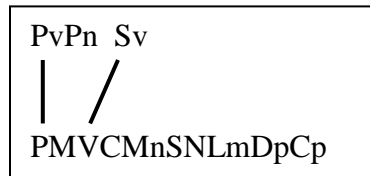


(2.2.1) $L_s =$; $R_M = VP$;

(2.2.2.1) $L_s =$; $R_M = VP$; $R_U = SvV$; $N = Sv$;

Ancestor (N) = $PMVCMnSNLmDpCp$; $L_e = SvV$

$SvV \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow SvV$

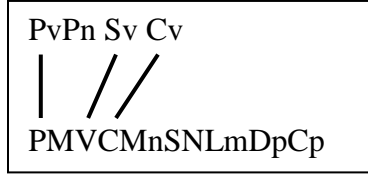


(2.2.1) $L_s =$; $R_M = CP$;

(2.2.2.1) $L_s =$; $R_M = CP$; $R_U = CvC$; $N = Cv$;

Ancestor (N) = $PMVCMnSNLmDpCp$; $L_e = CvC$

$CvC \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow CvC$



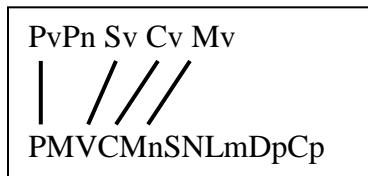
(2.2.1) $L_s = ; R_M = MnM;$

(2.2.2.1) $L_s = ; R_M = MnM; R_U = MvMn; N = Mv;$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = MvMn$

$MvMn \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow \setminus$

$MvMn$

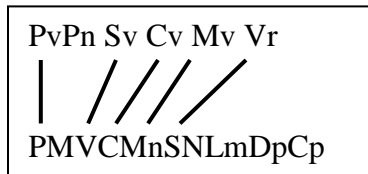


(2.2.1) $L_s = ; R_M = SM;$

(2.2.2.1) $L_s = ; R_M = SM; R_U = VrS; N = Vr;$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = VrS$

$VrS \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow VrS$

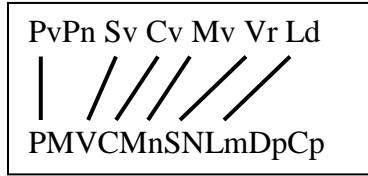


(2.2.1) $L_s = ; R_M = DpP;$

(2.2.2.1) $L_s = ; R_M = DpP; R_U = LdDp; N = Ld;$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = LdDp$

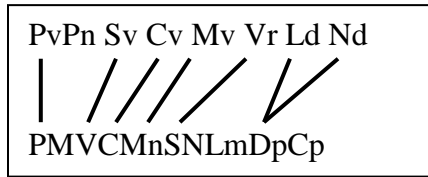
$LdDp \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow LdDp$



(2.2.2.1) $L_s = ; R_M = DpP; R_U = NdDp; N = Nd;$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = NdDp$

$NdDp \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow NdDp$

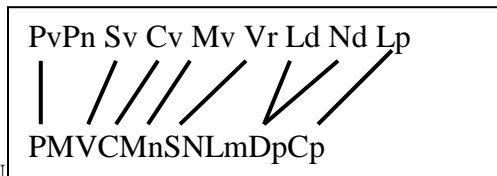


(2.2.1) $L_s = ; R_M = CpP;$

(2.2.2.1) $L_s = ; R_M = CpP; R_U = LpCp; N = Lp;$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = LpCp$

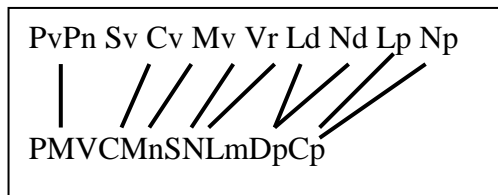
$LpCp \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow LpCp$



(2.2.2.1) $L_s = ; R_M = CpP; R_U =$

Ancestor (N) = MVCMnSNLmDpCp; $L_e = NpCp$

$NpCp \rightarrow PMVCMnSNLmDpCp$ and $PMVCMnSNLmDpCp \rightarrow NpCp$



From applying the formal technique to KFP domain yield a nested relational schema $NT1(PMVCMnSNLmDpCp (Sv)^*(PvPn)^* (Cv)^* (Vr)^* (Mv)^*(Ld)^*(Nd)^*(Lp)^*(Np)^*$). This nested relation is then translated into SQL: 2003 schema in following section.

3.4 Resulting Schemas

The resulting SQL:2003 schemas yield from applying Methodology 1 and Methodology 2 is shown in Table 12 and Table 13 respectively. The schemas from Table 12 was created by looking at the extended class of Figure 5, e.g. linear discrete process had one attribute therefore a type is created with such an attribute. Class such as Process model was extended from the original class of Figure 4. Attribute such as `<<array>> MModel_T: (<<ref>> Measurementmodel)` is defined as a multiset as shown in Table 12.

Methodology2 schemas were based on Figure 4 class diagram, where types are created for each class of the KFP class diagram. Components are created for classes showing classes that are linked to a class e.g. Sva_component refer to state variable which contains the component state vector and state variable. The result from applying algorithm B in Table 10 shows one nested table. This information is used to create a nested table in Table 13.

Table 12: SQL:2003 schema drawn using Methodology I

<pre> CREATE TYPE linearDiscreteProcess under discreteProcess as (linear Boolean); CREATE TYPE nonlinearDiscreteProcess under discreteProcess as (linear boolean); CREATE TYPE discreteProcess as under processModel as (equation string, ldp ref(linearDiscreteProcess), ndp ref(nonlinearDiscreteProcess)); CREATE TYPE ProcessModel as (equation string, linear boolean, timeVariant boolean, valueType String, mmodel_t ref(MeasurementModel) MULTISET, svt ref(StateVector), cpt ref(ContinuosProcess), dpt ref (DiscreteProcess)); CREATE TYPE stateVector as (Pmodel_t ref(ProcessModel) MULTISET, Svar ref(StateVariable) MULTISET); CREATE TYPE statevariable as (value Real, svect ref(StateVector) MULTISET); CREATE TYPE processNoise as (Pmodel_t ref(ProcessModel) MULTISET, pvar ref(processVariable) MULTISET); CREATE TYPE processVariable as (mean Real, noise Real, value Real, pnoise ref(processNoise) MULTISET); CREATE TYPE controlVector as (timevariant boolean, Pmodel_t ref(ProcessModel) MULTISET, cvar ref(controlVariable) MULTISET); CREATE TYPE controlVariable as (value real, cvec ref(controlVector) MULTISET); </pre>	<pre> CREATE TYPE continuousProcess under ProcessModel as (equation Real, lcp ref(linearcontinuousProcess), ncp ref(nonlinearContinuousProcess)); CREATE TYPE linearContinuousProcess under continuousProcess as (linear boolean); CREATE TYPE nonLinearContinuousProcess under continuousProcess as (linear boolean); CREATE TYPE measurementModel as (equation string, linear boolean, timeVariant Boolean, valueType string, Pmodel_t ref(ProcessModel) MULTISET); CREATE TYPE linearMeasurement under measurementModel as (linear boolean); CREATE TYPE nonLinearMeasurement under measurementModel as (linear boolean); CREATE TYPE MeastateVector as (mmodel_t ref(MeasurementModel) MULTISET, msvar ref(meastateVariable) MULTISET); CREATE TYPE MeastateVariable as (value Real, msVec ref(meastateVector) MULTISET); CREATE TYPE measurementNoise as (mmodel_t ref(MeasurementModel) MULTISET, mVar ref(measurementVariable) MULTISET); CREATE TYPE measurementVariable as (Mean Real, value Real, noise String, mNoise ref(measurementNoise) </pre>
---	--

MULTISET);

CREATE TABLE LDiscreteProcess of
linearDiscreteProcess;

CREATE TABLE NonlDiscreteProcess of
nonlinearDiscreteProcess;

CREATE TABLE PModel of ProcessModel;

CREATE TABLE SVector of stateVector;

CREATE TABLE Svariable of statevariable;

CREATE TABLE PNoise of processNoise;

CREATE TABLE PVariable of
processVariable;

CREATE TABLE CVector of controlVector;

CREATE TABLE cVariable of
controlVariable;

CREATE TABLE LContinuousProcess of
linearContinuousProcess;

CREATE TABLE NLContinuousProcess of
nonLinearContinuousProcess;

CREATE TABLE MModel of
measurementModel;

CREATE TABLE LMeasurement of
linearMeasurement;

CREATE TABLE NLMeasurement of
nonLinearMeasurement;

CREATE TABLE MSVector of
meastateVector;

CREATE TABLE MSVariable of
meastateVariable;

CREATE TABLE MNoise of
measurementNoise;

CREATE TABLE MVariable of
measurementVariable;

Table 13: SQL: 2003 schema drawn using Methodology 2

<pre> CREATE TYPE linearDiscreteProcess under DProcess as(linear Boolean); CREATE TYPE nonlinearDiscreteProcess under DProcess as(linear boolean); CREATE TYPE discreteProcess under ProcessModel as (equation string); CREATE TYPE ProcessModel as (equation string, linear boolean, timeVariant boolean, valueType String); CREATE TYPE StateVector as (value Real); CREATE TYPE StateVariable as (equation string); CREATE TYPE ProcessNoise as (equation string); CREATE TYPE ProcessVariable as (mean Real, noise Real, value Real); CREATE TYPE ControlVector as (timevariant time); CREATE TYPE ControlVariable as (equation string); CREATE TYPE ContinuousProcess under ProcessModel as (equation Real); CREATE TYPE linearContinuousProcess under ContinuousProcess as (linear Boolean); CREATE TYPE nonLinearContinuousProcess under ContinuousProcess as (</pre>	<pre> linear boolean); CREATE TYPE measurementModel as (linear Boolean, timeVariant Boolean, valueType String); CREATE TYPE linearMeasurement under MeasurementModel as (linear Boolean); CREATE TYPE nonLinearMeasurement under MeasurementModel as (linear boolean); CREATE TYPE MstateVector as (equation string); CREATE TYPE MstateVariable as (value Real); CREATE TYPE measurementNoise as (equation string); CREATE TYPE measurementVariable as (Mean Real, value Real, noise String); Create TYPE Sva_component as(statevec StateVector MULTISSET, statevar StateVariable); Create TYPE Pn_component as (pronoise ProcessNoise MULTISSET, provar ProcessVariable); Create TYPE Cv_component as (convec ControlVector MULTISSET, contvar ControlVariable); Create TYPE Msv_component as(meastvec MeasurementStateVector MULTISSET, meassvar MeasurementStateVariable); Create TYPE PM_component as (promodel ProcessModel MULTISSET, statevec StateVector MULTISSET, </pre>
---	---


```
pronoise ProcessNoise MULTISSET,  
convec ControlVector MULTISSET);
```

```
Create TYPE MM_component as (  
meamod Measurement Model MULTISSET,  
meastvec MeasurementStateVector  
MULTISSET,  
meanoi MeasurementNoise MULTISSET,  
procesmo PM_component MULTISSET);
```

```
Create TYPE Mn_component as (  
meanoi MeasurementNoise MULTISSET,  
mvariable MeasurementVariable,  
cv_com ref (cv_component),  
mm_com ref (mm_component),  
meas_com ref (msv_component),  
pmod_com ref (pm_component),  
pn_com ref (pn_component),  
sva_com ref (sva_component));
```

```
Create TABLE Kalman_Filter of  
MN_component;
```

CHAPTER 4

ANALYSIS

With the use of Oracle 11g object-relational database management system to convert SQL:2003 schemas the analysis of our Kalman Filter Program was able to be achieved. Following the Oracle 10g SQL statements syntax in table 13 from [7] which is similar to that of Oracle 11g syntax [18] the requires types and tables are able to created. A more comprehensive syntax diagram is presented in Appendix E.

Table 14: Oracle 11g SQL syntax

```
CREATE [OR REPLACE] TYPE <type_name>
{ {IS | AS } OBJECT
| UNDER < supertype> }
{(attribute datatype,...)}
[[NOT] FINAL]
[[ NOT] INSTANTIABLE ];

ALTER TYPE type_name COMPILE;

CREATE TABLE table OF object_type
  NESTED TABLE nested_column STORE AS storage_table
```

Samples of the Oracle 11g SQL schemas are provided in Table 15 resulted from Table 12 SQL:2003 schemas. Classes such as ControlVariable, ControlVector and ProcessModel are presented in this table while the rest of the classes may be found in Appendix A. Abbreviations of classes name are implemented for simple reference in Table 16. Based on stereotype in Table 8, ControlVariable is an <<Object Type>>, therefore a table cvariable of ControlVaribale. While creating type ControlVariable no error occurred while compiling this type in Oracle based on no reference to another object type.

Table 15 : Methodology1 Oracle 11g SQL schema

<pre> Create or Replace TYPE cvariable AS object(cvequation varchar(30)); Create or Replace TYPE cvariable_tt AS TABLE OF REF cvariable; Create or Replace TYPE cvector AS object(ctimeVariant number(1)) cvar cvariable_tt)NOT FINAL;</pre>	<pre> Create or Replace TYPE cvector_tt AS TABLE OF REF cvector; Create or Replace TYPE pmodel AS object(pmid varchar(10), pequation varchar(30), plinear number(1), ptimeVariant number(1), pvalueType varchar(15), kf kfilter, cv cvector_tt, pn pnoise_tt, sv svector_tt)NOT FINAL;</pre>
--	--

Figure 18 provides views of tables from the schema given in Appendix A using Methodology I. The use of arrow lines from one table to another represents references to objects. The table svector contains the attributes svtime, pmode and svequation. Attribute svtime stores the time of the state vector. Attribute pmode stores the reference to the Process Model table object. Attribute svrowval and svcval stores the row value and the column value for the Kalman Filter Program respectively.

Table PModel_tb from Figure 10 contains the attributes, pmid, pequation, plinear, ptimevariant, pvalueType, kf, cv, pn and sv. The attribute pmid contains an id for pmodel_tb; pequation stores the process equation for the Kalman Filter; ptimevariant attributes keeps track of time for the Kalman Filter to be updated.

The table Kf in Figure 10 is created to join the process model with the measurement model in order to represent the class diagram in Oracle 11g. Kf contains only attribute is id which stores a unique value for each process.

In handling Generalization/ specialization an approach is adopted from that of the work of R. Chennamaneni [7]. However, due to the uniqueness of the KFP class diagram, where some of the subclasses such as discrete process and continuous process are also superclass for other objects, tables are created for each superclass.

Table 16: Abbreviation of Class Name

Abbreviation	Class Name
pmodel	Process Model
cvector	Control Vector
cvariable	Control Variable
pvariable	Process Variable
pnose	Process Noise
svariable	State Variable
svector	State Vector
dprocess	Discrete Process
cprocess	Continuous Process
ldiscretep	Linear Discrete Process
nldiscretep	Nonlinear Discrete Process
nlcprocess	Nonlinear Continuous Process
Abbreviation	Class Name
lcprocess	Linear Continuous Process
mmodel	Measurment Model

mvariable	Measurement Variable
mnoise	Measurement Noise
msvariable	State Variable
msvector	Measurement Vector
nlmeas	NonLinear Measurement
lmeas	Linear Measurement

Table: SVector

Svtime	Pmode	Svequation

Table: PModel_tb

Pmid	Pequation	Plinear	Ptimevariant	pvalueType	Kf	Cv	Pn	sv

Table: Kf

id

Table: cv

ctimeVariant	Cvar

Figure 10: Partial Table structure using Methodology I

4.2 Methodology 2

Table 17 shows sample of Oracle 11g SQL schema for the SQL: 2003 schema of Table 12. A detail representation of Oracle 11g can be found in Appendix B. Table 18 shows the single nested normal form table, which was produced from the Kalman Filter Program using Methodology 2. The Mn_Component type contains user define attributes and collections such as meanoi, mvariable, cv_com, mm_com, meas_com, pmod_com, pn_com and sva_com. Meanoi is of MNoise_tt collection(nested table) type. The attribute mvariable is of data type msvar. Cv_com is of Cv_component which is a collection type. The attribute type mm_com represent the collection of mm_component.

Table 17: Methodology2 Oracle 11g SQL schema

<pre> Create or Replace TYPE cvar as object(cvequation varchar(30)); CREATE OR REPLACE TYPE cvar_tt AS TABLE OF cvar; Create or Replace TYPE cvec AS object(ctimeVariant number(1))NOT FINAL; Create or Replace TYPE pmodel AS object(pequation varchar(30), plinear number(1), ptimeVariant number(1), pvalueType varchar(15), pm procmeas)NOT FINAL; </pre>	<pre> Create or Replace TYPE Mn_Component AS Object(meanoi mnoise_tt, mvariable msvar, cv_com ref cv_component, mm_com ref mm_component, meas_com ref msv_component, pmod_com ref pm_component, pn_com ref pn_component, sva_com ref sva_component)Not Final; CREATE TABLE Kalman_Filter of Mn_Component NESTED TABLE measnoi STORE AS KF_NT;; </pre>
---	--

Meas_com attribute contains the collection of msv_component. The attribute Pmod_com represents the collection of Pm_component and pn_com represents the collection of pn_component. Attribute sva_com, represents the collection sva_component.

Table 18: Kalman_Filter

Attribute Name	Data Type		
MEANOI	(row, col) <table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="width: 40px; height: 15px;"></td> <td style="width: 40px; height: 15px;"></td> </tr> </table>		
MVARIABLE	(mean, value, noise)		
CV_COM	→		
MM_COM	→		
MEAS_COM	→		
PMOD_COM	→		
PN_COM	→		
SVA_COM	→		

4.3 Comparison of Methodologies

A similar quantitative comparison is carried out as that of the previous research. The results can be found in Table 19, where the SQL:2003 schema from Table 12 and Table 13 form the case study class diagram of Figure 4 and the Oracle implementation specification from Appendix A and Appendix B are compared.

Examining the base table from Table 19 shows the same number of base type. The number of intermediate types is zero, due to no cycle being reduced from the algorithm in Methodology 2. More tables were derived from using Methodology 1 as compared to Methodology 2, this is due to the modeling nature of the formal method where complex representations are created, thus fewer tables while Methodology 1 create simple tables which results in more tables being created. Methodology 2 produced reference type attributes; the ratio of reference table was 1:4 of Methodology 2 to Methodology 1. This was based on the overlapping principle of the Methodology 1

where attributes are reused in classes, while Methodology 2 model classes with their unique attributes.

From the oracle implementation specific types, the result in Table 19 shows a greater number of oracle implementation types from Methodology 2 as compared with Methodology 1. This was resulted from the additional use of component types to group individual component, in Methodology 1 reference is primarily used to carry out this feature while in Methodology 2; modeling Oracle table is geared more towards using nested table.

With the a absence of disjoint relations, Methodology 2 produces 1 nested while Methodology 1 extensive use of UML, transform each <<Object Type>> stereotyped class as a table which resulted in a greater number of nested table.

Table 19: Quantitative comparison

	Methodology 1	Methodology2
Number of base level types	20	20
Number of intermediate types	0	0
Number of tables	18	1
Number of reference type attributes	24	6
Oracle implementation specific types	10	15
Oracle implementation specific storage nested tables	7	1

4.4 Evaluation of Class Diagrams

With the use of a Kalman Filter Program (KFP) class diagram, both the formal and informal methodologies were stress tested. Both methodologies were designed to be test on familiar system, however this case study prove that both methodologies are able to convert various class diagram. The KFP class diagram carried similar classification of relationship as Airline Flight Reservations (AFR) class diagram, such as association and generalization.

Aggregation relationships were extensively used in KFP while none is found in the AFR. With this difference, the transformation process treated it as a semi-directional relationship. The number of aggregation was quite often used in the modeling class diagram of the KFP. The KFP contains one association which in compared to the AFR, where it is extensively used.

Generalization relationship was found more in the modeling process of the KFP as compared to the AFR. Generalization is treated as a semi-direction specialization which does not influence the class which the specialization arrow is pointing to.

4.5 Evaluation of Methodologies

Methodology 1 and Methodology 2 applied different transformation techniques to the KFP which yield SQL statements which were able to translate into Oracle 11g format. Each methodology introduced different transformation process. Methodology 1 requires fewer steps than that of Methodology 2 to transform KFP class diagram into SQL statements. In order to transform KFP class diagram extensive knowledge of UML is

required. Methodology 2 requires an extensive mathematical approach in order to carry out the transformation process.

Methodology 1 retains most of the class structure which makes reverse engineering easy to carry out. Tables derived from Methodology 2 are more complex and are fewer. Tables derived from Methodology 1 are simple in nature and are numerous. Methodology 1 is suited for modeling class diagrams with simple structure. While Methodology 2 suited for modeling class diagram with complex classes such as cycles.

Methodology 1 carries a high overhead due to the increase in number of tables and reference types being produced, however multiple accesses of relations are possible.

4.6 Similarity and Difference of KFP and AFR Methodologies

4.6.1 Methodology 1

Looking on the overall research result from the work of R. Chennamaneni [7] which looks at the AFR and the present case study, KFP shows similar result from the transformation process. A comparison of Methodology 1 is shown in Table 20 of both case studies. In examining each element, the base level would unlikely to be the same base on the notion of modeling different domain. The base level type is based on the number of classes in the class diagram.

No intermediate type occurred in Methodology1 of both domains, due to the direct transformation of classes to tables. Due to the generalization/ specialization in KFP the number of tables shows a decrease in the number compared to AFR number of tables. This result was based on the nature of the domain, as in the case of the AFR domain, some attributes were linked to separate tables while in the KFP each attributes

were linked to one large table. The number of reference types showed similar results. This was based on the interconnectivity of the class diagram in each domain.

Table 20: Comparison of Methodology 1

	KFP Methodology 1	AFR Methodology1
Number of base level types	21	16
Number of intermediate types	0	0
Number of tables	9	14
Number of reference type attributes	24	25
Oracle implementation specific types	10	7
Oracle implementation specific storage nested tables	7	13

With an increase in the number of base type, naturally there would be an increase in the Oracle implementation type of the SQL: 2003 statements. The Oracle implementation specific storage nested tables shows a different trend base on the previous explanation. The KFP had a smaller number of these tables compared to the numbers found in AFR. This was trace to the interconnectivity of the class diagram, in AFR domain some element may work independently, while in the KFP each element is interrelated.

4.6.2 Methodology 2

Examining the results in Table 21 about the comparison of Methodology 2 for the KFP and the AFR system, the KFP contains more base level types than that of AFR. This is based on the uniqueness of a domain which results in various numbers of base level types. No intermediate types were found in KFP Methodology 2, this is based on the absence of role names or association names between classes. However, in AFR role names or association are found which resulted in the number of intermediate types. Also

sub-classes are introduced in AFR domain to remove cycle, whereas in the KFP no cycle existed.

Table 21: Comparison of Methodology 2

	KFP Methodology2	AFR Methodology2
Number of base level types	21	16
Number of intermediate types	0	5
Number of tables	1	4
Number of reference type attributes	6	0
Oracle implementation specific types	15	5
Oracle implementation specific storage nested tables	1	4

A higher number of tables were found in AFR compared to that of the KFP; this is due to the use of the algorithm to remove semantically overloaded elements which resulted in separation of classes forming specialization. An occurrence of the reference types was found in the KFP where none was found in the AFR. In the KFP, in order to create a nested table from the class diagram which had components isolated, reference was used in order to join these separate objects.

With an increase in the number of base type, an increase will occur in the Oracle implementation specific types. This was the case for the KFP. The algorithm to remove cycle was not applied to the KFP, thus the result shows no separation of attributes from the main class. While cycle existed in the AFR domain, therefore the result from the algorithm shows individual tables which are form from the removal of cycles.

4.7 Comparison of Conclusions

Based on the conclusion reached from the transformation process of the KFP similarities are seen in the work done by the previous work of R. Chennamaneni [7] which looked at the AFR system. Both class diagrams produced fewer steps from Methodology 1 than of Methodology 2. Methodology 1 maintains the structure of both KFP and the AFR which makes reverse engineering possible. Methodology 1 produced more tables with simple structure for both class diagrams.

Methodology 2 produced fewer tables which were complex for both class diagrams. A simple requirement is noted for both systems, involving extensive knowledge of UML is required to use Methodology 1 while knowledge of Mathematics is required to carry out the transformation process of Methodology 2.

CHAPTER 5

CONCLUSION

A comparative analysis was carried out to validate the benefits of formal versus informal software transformation. The transformation techniques were applied to a Kalman Filter Program class diagram. The first aspect of the transformation process, transformed the KFP class diagram to SQL: 2003 statements which are presented in Chapter 3. The schemas from this transformation process were later converted to Oracle 11g schema which can be found in Appendix A and Appendix B.

Based on the transformation process, a focus on whether this transformation process is able to validate Chennamaneni's work will be examined in this chapter.

5.1: Validation Process

The goal of this research is to validate the work carried out by Chennamaneni [7]. A KFP class diagram was used to carry out a transformation process, based on the results the transformation process was able to transform the class diagram into to SQL statements. This is shown in Appendix A and B the transformation results from applying both methodologies to the KFP compared to the accomplishment SQL statements which were derived from applying both methodologies as well to the AFR system.

The complexity of relationship among the class diagrams being used is another area to note. In the AFR system, generalization and association classification of

relationship existed, while in KFP association, aggregation and generalization were evident. This presents the uniqueness of the class diagrams, which proves that the methodologies are able to transform various types of class diagram relationship.

A quantitative comparison of the methodologies used was carried out on each class diagram. The results of KFP showed in Table 19, similarities to the result from AFR shown in Table 4. The differences are explained in section 4.6 which overall shows that the quantitative comparison produces similar results which substantiate the work done by Rajan's.

Similar results were reached when examining the advantages and disadvantages from of the Methodologies used to transform KFP and AFR system. These include fewer steps being used by a particular Methodology; another result showed that fewer tables were achieved by a particular Methodology. Other results reached are outlined in Section 4.7.

With these results reached from the transformation process, it gives validation to transformation process being used by Rajan's. With the transformation process being validated future steps in this area of research can be further pursue which are outlined in the Section 5.2.

5.2 Future Work

The automation of the transformation process could be pursued, which would aid in the complexity of the transformation process. This would remove the possibility of human error and facilitate a standard approach to transform different problem domain

class diagrams. With the completion of static analysis, a dynamic approach of this research is required. Another area of future work is to look at the performance of a system using a formal approach to model a domain and an informal approach. Within this scope of work, running the system on sequential system and also a parallel system would provide recommendations to developers who are modeling systems.

Appendix A

The results from Oracle 11g SQL schema of Methodology 1

```
Create or Replace TYPE  
kfilter AS object( id varchar(30));
```

```
Create or Replace TYPE  
pmodel AS object(  
pmid varchar(10),  
pequation varchar(30),  
plinear number(1),  
ptimeVariant number(1),  
pvalueType varchar(15),  
kf kfilter,  
cv cvector,  
pn pnoise,  
sv svector)NOT FINAL;
```

```
Create or Replace TYPE  
pmodel_tt AS TABLE of REF pmodel;
```

```
Create or Replace TYPE  
cvariable as object(  
cvequation varchar(30));
```

```
Create or ReplaceTYPE  
cvariable_tt AS TABLE of REF cvariable;
```

```
Create or Replace TYPE  
cvector AS object(  
ctimeVariant number(1),  
cvar cvariable_tt)NOT FINAL;
```

```
Create or Replace TYPE  
cvector_tt AS TABLE of REF cvector;
```

```
Create or Replace TYPE  
pvariable as object(  
pmean real,  
pnoise real,  
pvalue REAL);  
Create or Replace TYPE  
pvariable_tt AS TABLE of REF pvariable;
```

```
Create or Replace TYPE
```

```
pnoise AS object(  
pntimeVariant number(1),  
pvar pvariable_tt)NOT FINAL;
```

```
Create or Replace TYPE  
pnoise_tt AS TABLE of REF pnoise;
```

```
Create or Replace TYPE  
svariable as object(  
svequation varchar(30));
```

```
Create or Replace TYPE  
svariable_tt AS TABLE of REF svariable;
```

```
Create or Replace TYPE  
svector AS object(  
svtimeVariant number(1),  
svar svariable_tt)NOT FINAL;
```

```
Create or Replace TYPE  
svector_tt AS TABLE of REF svector;
```

```
Create or Replace TYPE  
dprocess UNDER pmodel (  
dpequation varchar(30) )NOT FINAL;
```

```
Create or Replace TYPE  
cprocess UNDER pmodel (  
cpequation varchar(30) )NOT FINAL;
```

```
Create or Replace TYPE  
ldiscretep UNDER dprocess (  
llinear NUMBER(1) )not final;
```

```
Create or Replace TYPE  
nldiscretep UNDER dprocess (  
nllinear NUMBER(1) )not final;
```

```
Create or Replace TYPE  
nlcprocess UNDER cprocess (  
nlclinear NUMBER(1) )not final;
```

```
Create or Replace TYPE
```

```

lprocess UNDER cprocess (
lclinear NUMBER(1) )not final;

Create or Replace TYPE
mmodel AS object(
mequation varchar(30),
mlinear number(1),
mtimeVariant number(1),
mvalueType varchar(15),
kft kfilter,
mn mnoise,
msv msvector)NOT FINAL;

Create or Replace TYPE
mmodel_tt AS TABLE of REF mmodel;

Create or Replace TYPE
mvariable AS object(
mmean real,
mmvalue real,
mnoise real);

Create or Replace TYPE
mvariable_tt AS TABLE of REF mvariable;

Create or Replace TYPE
mnoise as object(
mnequation varchar(30),
mvar mvariable_tt)NOT FINAL;

Create or Replace TYPE
mnoise_tt AS TABLE of REF mnoise;

Create or Replace TYPE
msvariable as object(
msvalue REAL );

Create or Replace TYPE
msvariable_tt AS TABLE of REF
msvariable;

Create or Replace TYPE
msvector as object(
msvequation varchar(30),
msvar msvariable_tt)NOT FINAL;

Create or Replace TYPE
msvector_tt AS TABLE of REF msvector;
Create or Replace TYPE
nlmeas under mmodel(

```

```

tlinear number(1)) not final;

Create or Replace TYPE
lmeas under mmodel(
llinear number(1)) not final;

CREATE TABLE PModel_tb OF PModel;

CREATE TABLE SVector_tb OF SVector
NESTED TABLE pmode STORE AS
pm_nt,
NESTED TABLE svar STORE AS svar_nt;

CREATE TABLE Svariable_tb of svariable
NESTED TABLE sv STORE AS svect_nt;

CREATE TABLE PNoise_tb of PNoise
NESTED TABLE pmo STORE AS pmd_nt,
NESTED TABLE pvar STORE AS pvar_nt;

CREATE TABLE PVariable_tb OF
PVariable
NESTED TABLE pn STORE AS pn_nt;

CREATE TABLE CVector_tb OF CVector
NESTED TABLE pmod STORE AS
pmo_nt,
NESTED TABLE cvar STORE AS cvar_nt;

CREATE TABLE cVariable_tb of
CVariable
NESTED TABLE cv STORE AS cv_nt;

CREATE TABLE MModel_tb of
MModel;

CREATE TABLE MSVector_t of
MSVector,
NESTED TABLE mmdl STORE AS
mm_nt,
NESTED TABLE msvar STORE AS
msvar_nt;

CREATE TABLE MNoise_t of MNoise,
NESTED TABLE mmd STORE AS mmdl_nt,
NESTED TABLE mvar STORE AS mvar_nt;
CREATE TABLE MSVariable_t of
MSVariable,
NESTED TABLE msv STORE AS mno_nt;

```

Appendix B

The results of Oracle 11g SQL schema of Methodology 2

```
Create or Replace type ProcMeas AS
object( id varchar(30));

Create or Replace
TYPE cvar as object(
cvalue real);

Create or Replace
TYPE pvar as object(
pmean real,
pnoise real,
pvalue real );

Create or Replace
TYPE svar as object(
svalue real);

Create or Replace
TYPE cvec AS object(
ctimeVariant number(1))NOT FINAL;

CREATE OR REPLACE TYPE cvec_tt AS
TABLE OF cvec;

Create or Replace
TYPE pnoise AS object(
pnrow REAL,
pncol real)NOT FINAL;

Create or Replace
TYPE pnoise_tt AS TABLE OF pnoise;

Create or Replace
TYPE svec AS object(
svtimeVariant number(1),
svrow real,
svcol real)NOT FINAL;

CREATE OR REPLACE TYPE svec_tt AS
TABLE OF svec;

Create or Replace
TYPE pmodel AS object(
```

```
pequation varchar(30),
plinear number(1),
ptimeVariant number(1),
pvalueType varchar(15),
pm procmeas)NOT FINAL;

Create or Replace
TYPE pmodel_tt AS TABLE OF pmodel;

Create or Replace TYPE
dpro UNDER pmodel (
dpequation varchar(30) )NOT FINAL;

Create or Replace TYPE
cpro UNDER pmodel (
cpequation varchar(30) )NOT FINAL;

Create or Replace TYPE
ldpro UNDER dpro (
llinear NUMBER(1) )not final;

Create or Replace TYPE
nldpro UNDER dpro (
nllinear NUMBER(1) )not final;

Create or Replace TYPE
lcpro UNDER cpro (
clinear NUMBER(1) )not final;

Create or Replace TYPE
nlcpro UNDER cpro (
nclinear NUMBER(1) )not final;

Create or Replace
TYPE mmodel AS object(
mequation varchar(30),
mlinear number(1),
mtimeVariant number(1),
mvalueType varchar(15),
pm procmeas)NOT FINAL;

Create or Replace
TYPE mmodel_tt AS TABLE OF mmodel;
```

```
Create or Replace TYPE  
nlmeas under mmodel(  
tlinear number(1)) not final;
```

```
Create or Replace type  
lmeas under mmodel(  
linear number(1)) not final;
```

```
Create or Replace  
TYPE msvar AS object(  
mmean real,  
mmvalue real,  
mnoise real);
```

```
CREATE OR REPLACE TYPE msvar_tt  
AS TABLE OF msvar;
```

```
Create or Replace  
TYPE mnoise as object(  
mnrow REAL,  
mncol real)NOT FINAL;
```

```
Create or Replace  
TYPE mnoise_tt AS TABLE OF mnoise;
```

```
Create or Replace  
TYPE measvar AS object(  
mvalue real);
```

```
Create or Replace  
TYPE msvec as object(  
msvrow real,  
msvcol real)NOT FINAL;
```

```
Create or Replace  
TYPE msvec_tt AS TABLE OF msvec;
```

```
Create or Replace  
TYPE Sva_Component AS Object(  
statevec svec_tt,  
statevar svar)Not Final;
```

```
Create or Replace  
TYPE Pn_Component AS Object(  

```

```
pronoise prnoise_tt,  
provar pvar)Not Final;
```

```
Create or Replace  
TYPE Cv_Component AS Object(  
convec cvec_tt,  
contvar cvar)Not Final;
```

```
Create or Replace  
TYPE Msv_Component AS Object(  
meastvec msvec_tt,  
meassvar measvar)Not Final;
```

```
Create or Replace  
TYPE Pm_Component AS Object(  
promodel pmodel_tt,  
statevec svec_tt,  
pronoise prnoise_tt,  
convec cvec_tt)Not Final;
```

```
Create or Replace  
TYPE MM_Component AS Object(  
meamod mmodel_tt,  
meastvec msvec_tt,  
meanoi mnoise_tt,  
procesmo Pm_component)Not Final;
```

```
Create or Replace  
TYPE Mn_Component AS Object(  
meanoi mnoise_tt,  
mvariable msvar,  
cv_com ref cv_component,  
mm_com ref mm_component,  
meas_com ref msv_component,  
pmod_com ref pm_component,  
pn_com ref pn_component,  
sva_com ref sva_component  
)Not Final;
```

```
CREATE TABLE Kalman_Filter of  
Mn_Component NESTED TABLE measnoi  
STORE AS KF_NT;
```

Appendix C

Schema execution details from Appendix A showing details about types and tables from Oracle 11g application of Methodology 1.

```
Create or Replace
TYPE kfilter AS object(
id varchar(30));

TYPE KFILTER compiled

Create or Replace
TYPE cvariable as object(
cvname varchar(30),
cvequation varchar(30),
cvrow real,
cvcol real);

TYPE CVARIABLE compiled

Create or Replace
TYPE cvariable_tt AS TABLE of REF
cvariable;

TYPE CVARIABLE_TT compiled

Create or Replace
TYPE cvector AS object(
ctimeVariant number(1),
cname varchar(30),
crow real,
ccol real,
cvar cvariable_tt)NOT FINAL;

TYPE CVECTOR compiled

create or replace
TYPE cvector_tt AS TABLE of REF cvector;

TYPE CVECTOR_TT compiled

Create or Replace
TYPE pvariable as object(
pmean real,
pnoise real,
pvalue real );

TYPE PVARIABLE compiled

Create or Replace
TYPE pvariable_tt AS TABLE of REF
pvariable;
```

```
TYPE PVARIABLE_TT compiled

Create or Replace
TYPE pnoise AS object(
pntimeVariant number(1),
pnname varchar(30),
pnrow REAL,
pncol real,
pvar pvariable_tt)NOT FINAL;

TYPE PNOISE compiled

Create or Replace
TYPE pnoise_tt AS TABLE of REF pnoise;

TYPE PNOISE_TT compiled

Create or Replace
TYPE svariable as object(
svname varchar(30),
svequation varchar(30),
svrowval REAL,
svcolval real);

TYPE SVARIABLE compiled

Create or Replace
TYPE svariable_tt AS TABLE of REF
svariable;

TYPE SVARIABLE_TT compiled

Create or Replace
TYPE svector AS object(
svtimeVariant number(1),
svname varchar(30),
svrowval REAL,
svcolval real,
svar svariable_tt)NOT FINAL;

TYPE SVECTOR compiled

Create or Replace
TYPE svector_tt AS TABLE of REF svector;

TYPE SVECTOR_TT compiled

Create or Replace
```

```

TYPE pmodel AS object(
  pmid varchar(10),
  pequation varchar(30),
  plinear number(1),
  ptimeVariant number(1),
  pvalueType varchar(15),
  kf kfilter,
  cv cvector_tt,
  pn pnoise_tt,
  sv svector_tt)NOT FINAL;

TYPE PMODEL compiled

Create or Replace TYPE
dprocess UNDER pmodel (
dpequation varchar(30) )NOT FINAL;

TYPE DPROCESS compiled

Create or Replace TYPE
cprocess UNDER pmodel (
cpequation varchar(30) )NOT FINAL;

TYPE CPROCESS compiled

Create or Replace TYPE
ldiscretep UNDER dprocess (
llinear NUMBER(1) )not final;

TYPE LDISCRETEP compiled

Create or Replace TYPE
nldiscretep UNDER dprocess (
nllinear NUMBER(1) )not final;

TYPE NLDISCRETEP compiled

Create or Replace TYPE
nlcprocess UNDER cprocess (
nlclinear NUMBER(1) )not final;

TYPE NLCPROCESS compiled

Create or Replace TYPE
lcprocess UNDER cprocess (
lclinear NUMBER(1) )not final;

TYPE LCPROCESS compiled

Create or Replace
TYPE mvariable AS object(
  mmean real,
  mmvalue real,
  mnoise real);

TYPE MVARIABLE compiled

```

```

Create or Replace
TYPE mvariable_tt AS TABLE of REF
mvariable;

TYPE MVARIABLE_TT compiled

Create or Replace
TYPE mnoise as object(
  mnname varchar(30),
  mnequation varchar(30),
  mnrow REAL,
  mncol real,
  mvar mvariable_tt)NOT FINAL;

TYPE MNOISE compiled

create or replace
TYPE mnoise_tt AS TABLE of REF mnoise;

TYPE MNOISE_TT compiled

Create or Replace
TYPE msvariable as object(
  msvalue REAL);

TYPE MSVARIABLE compiled

Create or Replace
TYPE msvariable_tt AS TABLE of REF
msvariable;

TYPE MSVARIABLE_TT compiled

Create or Replace
TYPE msvector as object(
  msvname varchar(30),
  msvequation varchar(30),
  msvrow REAL,
  msvcol real,
  msvar msvariable_tt)NOT FINAL;

TYPE MSVECTOR compiled

Create or Replace
TYPE msvector_tt AS TABLE of REF
msvector;

TYPE MSVECTOR_TT compiled

Create or Replace
TYPE mmodel AS object(
  mequation varchar(30),
  mlinear number(1),
  mtimeVariant number(1),
  mvalueType varchar(15),

```

```

kft kfilter,
mn mnoise_tt,
msv msvector_tt)NOT FINAL;

TYPE MMODEL compiled

Create or Replace type
nlmeas under mmodel(
tlinear number(1)) not final;

TYPE NLMEAS compiled

Create or Replace type
lmeas under mmodel(
llinear number(1)) not final;

TYPE LMEAS compiled

alter type Pmodel compile;
alter type svector compile;
alter type cvector compile;
alter type pnoise compile;
alter type dprocess compile;
alter type cprocess compile;

CREATE TABLE SVector_tb OF SVector
NESTED TABLE svar STORE AS svar_nt;

CREATE TABLE succeeded.

CREATE TABLE Svariable_tb of svariable;

CREATE TABLE succeeded.

```

```

CREATE TABLE PNoise_tb of PNoise
NESTED TABLE pvar STORE AS pvar_nt;

CREATE TABLE succeeded.

CREATE TABLE PVariable_tb OF PVariable ;

CREATE TABLE succeeded.

CREATE TABLE CVector_tb OF CVector
NESTED TABLE cvar STORE AS cvar_nt;

CREATE TABLE succeeded.

CREATE TABLE cVariable_tb of CVariable ;

CREATE TABLE succeeded.

CREATE TABLE MSVector_t of
MSVector
NESTED TABLE msvar STORE AS msvar_nt;

CREATE TABLE succeeded.

CREATE TABLE MNoise_t of MNoise
NESTED TABLE mvar STORE AS mvar_nt;

CREATE TABLE succeeded.

CREATE TABLE MSVariable_t of MSVariable;

CREATE TABLE succeeded.

```

```

SQL> desc cprocess;
cprocess NOT FINAL

```

Name	Null?	Type
cpequation		varchar(30)

```

SQL> desc CVARIABLE
CVARIABLE NOT FINAL

```

Name	Null?	Type
cvname		varchar(30)
cvequation		varchar(30)
cvrow		real
cvcol		real

```

SQL> desc CVARIABLE_TT;
CVARIABLE_TT TABLE OF CVARIABLE

```

SQL> desc CVECTOR;
 CVECTOR NOT FINAL

Name	Null?	Type
ctimeVariant		number(1)
cname		varchar(30)
crow		REAL
ccol		real
cvar		cvariable_tt

SQL> desc CVECTOR_TT;
 CVECTOR_TT TABLE OF CVECTOR

SQL> desc DPROCESS;
 DPROCESS NOT FINAL

Name	Null?	Type
dpequation		varchar(30)

SQL> desc KFILTER;
 KFILTER is NOT FINAL

Name	Null?	Type
id		varchar(30)

SQL> desc LCPROCESS;
 LCPROCESS is NOT FINAL

Name	Null?	Type
lclinear		NUMBER(1)

SQL> desc LDISCRETEP;
 LDISCRETEP is NOT FINAL

Name	Null?	Type
llinear		NUMBER(1)

SQL> desc LMEAS;
 LMEAS is NOT FINAL

Name	Null?	Type
llinear		NUMBER(1)

SQL> desc MMODEL;
 MMODEL is NOT FINAL

Name	Null?	Type
mequation		varchar(30),
mlinear		number(1),
mtimeVariant		number(1),
mvalueType		varchar(15),
kft		kfilter,
mn		mnoise_tt,
msv		msvector_tt

SQL> desc MNOISE;
MNOISE is NOT FINAL

Name	Null?	Type
mname		varchar(30),
mnequation		varchar(30),
mrow		REAL,
mncol		real,
mvar		mvariable_tt

SQL> desc MNOISE_TT;
MNOISE_TT TABLE OF MNOISE

SQL> desc MSVARIABLE;
MSVARIABLE is NOT FINAL

Name	Null?	Type
msvalue		REAL

SQL> desc MSVARIABLE_TT;
MSVARIABLE_TT TABLE OF MSVARIABLE

SQL> desc MSVECTOR;
MSVECTOR is NOT FINAL

Name	Null?	Type
msvname		varchar(30),
msvequation		varchar(30),
msvrow		REAL,
msvcol		real,
msvar		msvariable_tt

SQL> desc MSVECTOR_TT;
MSVECTOR_TT TABLE OF MSVECTOR

SQL> desc MVARIABLE;
MVARIABLE is NOT FINAL

Name	Null?	Type
mmean		real,
mmvalue		real,
mnoise		real

SQL> desc MVARIABLE_TT;
MVARIABLE_TT TABLE OF MVARIABLE

SQL> desc NLCPROCESS;
NLCPROCESS is NOT FINAL

Name	Null?	Type
nlcllinear		NUMBER(1)

SQL> desc NLDISCRETEP;
NLDISCRETEP is NOT FINAL

Name	Null?	Type
------	-------	------

nlinear NUMBER(1)

SQL> desc NLMEAS;
NLMEAS is NOT FINAL

Name	Null?	Type
-----	-----	-----
tlinear		NUMBER(1)

SQL> desc PMODEL;
PMODEL is NOT FINAL

Name	Null?	Type
-----	-----	-----
pmid		varchar(10),
pequation		varchar(30),
plinear		number(1),
ptimeVariant		number(1),
pvalueType		varchar(15),
kf		kfilter,
cv		cvector_tt,
pn		pnoise_tt,
sv		svector_tt

SQL> desc PNOISE;
PNOISE is NOT FINAL

Name	Null?	Type
-----	-----	-----
pntimeVariant		number(1),
pname		varchar(30),
pnrow		REAL,
pncol		real,
pvar		pvariable_tt

SQL> desc PNOISE_TT;
PNOISE_TT TABLE OF PNOISE

SQL> desc PARIABLE;
PARIABLE is NOT FINAL

Name	Null?	Type
-----	-----	-----
pmean		real,
pnoise		real,
pvalue		REAL

SQL> desc PARIABLE_TT;
PARIABLE_TT TABLE OF PARIABLE

SQL> desc SVARIABLE;
SVARIABLE is NOT FINAL

Name	Null?	Type
-----	-----	-----
svname		varchar(30),
svequation		varchar(30),
svrowval		REAL,
svcolval		real

```
SQL> desc SVARIABLE_TT;
SVARIABLE_TT TABLE OF SVARIABLE
```

```
SQL> desc SVECTOR;
SVECTOR is NOT FINAL
```

Name	Null?	Type
svtimeVariant		number(1),
svname		varchar(30),
svrowval		REAL,
svcolval		real,
svar		svariable_tt

```
SQL> desc SVECTOR_TT;
SVECTOR_TT TABLE OF SVECTOR
```

```
SQL> select object_name, object_type, status from user_objects where object_type ='TYPE';
```

OBJECT_NAME	OBJECT_TYPE	STATUS
SVECTOR_TT	TYPE	VALID
SVECTOR	TYPE	VALID
SVARIABLE_TT	TYPE	VALID
SVARIABLE	TYPE	VALID
PVARIABLE_TT	TYPE	VALID
PVARIABLE	TYPE	VALID
PNOISE_TT	TYPE	VALID
PNOISE	TYPE	VALID
PMODEL	TYPE	VALID
NLMEAS	TYPE	VALID
NLDISCRETEP	TYPE	VALID
NLCPROCESS	TYPE	VALID
MVARIABLE_TT	TYPE	VALID
MVARIABLE	TYPE	VALID
MSVECTOR_TT	TYPE	VALID
MSVECTOR	TYPE	VALID
MSVARIABLE_TT	TYPE	VALID
MSVARIABLE	TYPE	VALID
MNOISE_TT	TYPE	VALID
MNOISE	TYPE	VALID
MMODEL	TYPE	VALID
LMEAS	TYPE	VALID
LDISCRETEP	TYPE	VALID
LCPROCESS	TYPE	VALID
KFILTER	TYPE	VALID
DPROCESS	TYPE	VALID
CVECTOR_TT	TYPE	VALID
CVECTOR	TYPE	VALID
CVARIABLE_TT	TYPE	VALID
CVARIABLE	TYPE	VALID
CPROCESS	TYPE	VALID

31 rows selected

SQL> select object_name, object_type, status from user_objects where object_type ='TABLE';

OBJECT_NAME	OBJECT_TYPE	STATUS
SVECTOR_TB	TABLE	VALID
SVAR_NT	TABLE	VALID
SVARIABLE_TB	TABLE	VALID
PVAR_NT	TABLE	VALID
PVARIABLE_TB	TABLE	VALID
PNOISE_TB	TABLE	VALID
MVAR_NT	TABLE	VALID
MSVECTOR_T	TABLE	VALID
MSVAR_NT	TABLE	VALID
MSVARIABLE_T	TABLE	VALID
MNOISE_T	TABLE	VALID
CVECTOR_TB	TABLE	VALID
CVAR_NT	TABLE	VALID
CVARIABLE_TB	TABLE	VALID

14 rows selected

SQL> desc CVAR_NT;

Name	Null	Type
COLUMN_VALUE		

SQL> desc CVARIABLE_TB

Name	Null	Type
CVNAME		VARCHAR2(30)
CVEQUATION		VARCHAR2(30)
CVROW		FLOAT(63)
CVCOL		FLOAT(63)

SQL> desc CVECTOR_TB;

Name	Null	Type
CTIMEVARIANT		NUMBER(1)
CNAME		VARCHAR2(30)
CROW		FLOAT(63)
CCOL		FLOAT(63)
CVAR		CVARIABLE_TT()

SQL> desc MNOISE_T;

Name	Null	Type
MNNAME		VARCHAR2(30)
MNEQUATION		VARCHAR2(30)
MNROW		FLOAT(63)
MNCOL		FLOAT(63)

MVAR

MVARIABLE_TT()

SQL> desc MSVAR_NT;

Name	Null	Type
COLUMN_VALUE		

SQL> desc MSVARIABLE_T;

Name	Null	Type
MSVALUE		FLOAT(63)

SQL> desc MSVECTOR_T;

Name	Null	Type
MSVNAME		VARCHAR2(30)
MSVEQUATION		VARCHAR2(30)
MSVROW		FLOAT(63)
MSVCOL		FLOAT(63)
MSVAR		MSVARIABLE_TT()

SQL> desc MVAR_NT;

Name	Null	Type
COLUMN_VALUE		

SQL> desc PNOISE_TB;

Name	Null	Type
PNTIMEVARIANT		NUMBER(1)
PNNAME		VARCHAR2(30)
PNROW		FLOAT(63)
PNCOL		FLOAT(63)
PVAR		PVARIABLE_TT()

SQL> desc PVAR_NT;

Name	Null	Type
COLUMN_VALUE		

SQL> desc PVARIABLE_TB;

Name	Null	Type
PMEAN		FLOAT(63)
PNOISE		FLOAT(63)
PVALUE		FLOAT(63)

```
SQL> desc SVAR_NT;
Name                Null                Type
-----
COLUMN_VALUE
```

```
SQL> desc SVARIABLE_TB;
Name                Null                Type
-----
SVNAME              VARCHAR2(30)
SVEQUATION          VARCHAR2(30)
SVROWVAL            FLOAT(63)
SVCOLVAL            FLOAT(63)
```

```
SQL> desc SVECTOR_TB;
Name                Null                Type
-----
SVTIMEVARIANT       NUMBER(1)
SVNAME              VARCHAR2(30)
SVROWVAL            FLOAT(63)
SVCOLVAL            FLOAT(63)
SVAR                SVARIABLE_TT()
```

Appendix D

Schema execution details from Appendix B showing details about types and tables from Oracle 11g application of Methodology 2.

<pre>Create or Replace type ProcMeas AS object(id varchar(30)); TYPE PROCMEAS compiled Create or Replace TYPE cvar as object(cvalue real); TYPE CVAR compiled Create or Replace TYPE pvar as object(pmean real, pnoise real, pvalue real); TYPE PVAR compiled Create or Replace TYPE svar as object(svalue real); TYPE SVAR compiled Create or Replace TYPE cvec AS object(ctimeVariant number(1))NOT FINAL; TYPE CVEC compiled CREATE OR REPLACE TYPE cvec_tt AS TABLE OF cvec; TYPE CVEC_TT compiled Create or Replace TYPE pnoise AS object(pnrow REAL, pncol real)NOT FINAL; TYPE PNOISE compiled create or replace TYPE pnoise_tt AS TABLE OF pnoise; TYPE pnoise_tt compiled Create or Replace TYPE svec AS object(svtimeVariant number(1), svrow real, svcol real)NOT FINAL;</pre>	<pre>TYPE SVEC compiled CREATE OR REPLACE TYPE svec_tt AS TABLE OF svec; TYPE SVEC_TT compiled Create or Replace TYPE pmodel AS object(pequation varchar(30), plinear number(1), ptimeVariant number(1), pvalueType varchar(15), pm_procmeas)NOT FINAL; TYPE PMODEL compiled create or replace TYPE pmodel_tt AS TABLE OF pmodel; TYPE pmodel_tt compiled Create or Replace TYPE dpro UNDER pmodel (dpequation varchar(30))NOT FINAL; TYPE DPRO compiled Create or Replace TYPE cpro UNDER pmodel (cpequation varchar(30))NOT FINAL; TYPE CPRO compiled Create or Replace TYPE ldpro UNDER dpro (llinear NUMBER(1))not final; TYPE LDPRO compiled Create or Replace TYPE nldpro UNDER dpro (nllinear NUMBER(1))not final; TYPE NLDPRO compiled Create or Replace TYPE lcpro UNDER cpro (clinear NUMBER(1))not final; TYPE LCPRO compiled</pre>
--	--

Create or Replace TYPE
nlcpro UNDER cpro (
nclinear NUMBER(1))not final;

TYPE NLCPRO compiled

Create or Replace
TYPE mmodel AS object(
mequation varchar(30),
mlinear number(1),
mtimeVariant number(1),
mvalueType varchar(15),
pm procmeas)NOT FINAL;

TYPE MMODEL compiled

create or replace
TYPE mmodel_tt AS TABLE OF mmodel;

TYPE MMODEL_TT compiled

Create or Replace TYPE
nlmeas under mmodel(
tlinear number(1)) not final;

TYPE NLMEAS compiled

Create or Replace type
lmeas under mmodel(
linear number(1)) not final;

TYPE LMEAS compiled

Create or Replace
TYPE msvar AS object(
mmean real,
mmvalue real,
mnoise real);

TYPE MSVAR compiled

CREATE OR REPLACE TYPE msvar_tt AS
TABLE OF msvar;

TYPE MSVAR_TT compiled

Create or Replace
TYPE mnoise as object(
mnrow REAL,
mncol real)NOT FINAL;

TYPE MNOISE compiled

create or replace
TYPE mnoise_tt AS TABLE OF mnoise;

TYPE MNOISE_TT compiled

Create or Replace
TYPE measvar AS object(
mvalue real);

TYPE MEASVAR compiled

Create or Replace
TYPE msvec as object(
msvrow real,
msvcol real)NOT FINAL;

TYPE MSVEC compiled

create or replace
TYPE msvec_tt AS TABLE OF msvec;

TYPE MSVEC_TT compiled

create or replace
TYPE Sva_Component AS Object(
statevec svec_tt,
statevar svar)Not Final;

TYPE SVA_COMPONENT compiled

create or replace
TYPE Pn_Component AS Object(
pronoise pnoise_tt,
provar pvar)Not Final;

TYPE PN_COMPONENT compiled

create or replace
TYPE Cv_Component AS Object(
convec cvec_tt,
contvar cvar)Not Final;

TYPE CV_COMPONENT compiled

create or replace
TYPE Msv_Component AS Object(
meastvec msvec_tt,
meassvar measvar)Not Final;

TYPE MSV_COMPONENT compiled

create or replace
TYPE Pm_Component AS Object(
promodel pmodel_tt,
statevec svec_tt,
pronoise pnoise_tt,
convec cvec_tt)Not Final;

TYPE PM_COMPONENT compiled

create or replace
TYPE MM_Component AS Object(
meamod mmodel_tt,
meastvec msvec_tt,
meanoi mnoise_tt,
procesmo Pm_component)Not Final;

TYPE MM_COMPONENT compiled


```

create or replace
TYPE Mn_Component AS Object(
meanoi mnoise_tt,
mvariable msvar,
cv_com ref cv_component,
mm_com ref mm_component,
meas_com ref msv_component,
pmod_com ref pm_component,
pn_com ref pn_component,
sva_com ref sva_component

```

```

)Not Final;
TYPE MN_COMPONENT compiled

CREATE TABLE Kalman_Filter of
Mn_Component
NESTED TABLE measnoi STORE AS KF_NT;

```

```
select object_name, object_type, status from user_objects where object_type ='TYPE';
```

OBJECT_NAME	OBJECT_TYPE	STATUS
SVEC_TT	TYPE	VALID
SVEC	TYPE	VALID
SVA_COMPONENT	TYPE	VALID
SVAR	TYPE	VALID
PVAR	TYPE	VALID
PROCMEAS	TYPE	VALID
PN_COMPONENT	TYPE	VALID
PNOISE_TT	TYPE	VALID
PNOISE	TYPE	VALID
PM_COMPONENT	TYPE	VALID
PMODEL_TT	TYPE	VALID
PMODEL	TYPE	VALID
NLMEAS	TYPE	VALID
NLDPRO	TYPE	VALID
NLCPRO	TYPE	VALID
MSV_COMPONENT	TYPE	VALID
MSVEC_TT	TYPE	VALID
MSVEC	TYPE	VALID
MSVAR_TT	TYPE	VALID
MSVAR	TYPE	VALID
MN_COMPONENT	TYPE	VALID
MNOISE_TT	TYPE	VALID
MNOISE	TYPE	VALID
MM_COMPONENT	TYPE	VALID
MMODEL_TT	TYPE	VALID
MMODEL	TYPE	VALID
MEASVAR	TYPE	VALID
LMEAS	TYPE	VALID
LDPRO	TYPE	VALID
LCPRO	TYPE	VALID
DPRO	TYPE	VALID
CV_COMPONENT	TYPE	VALID
CVEC_TT	TYPE	VALID
CVEC	TYPE	VALID
CVAR	TYPE	VALID
CPRO	TYPE	VALID

36 rows selected

```
SQL> desc CPRO;
CPRO extends METH2.PMODEL
```

CPRO is NOT FINAL NAME	NULL	TYPE

cpequation varchar(30)		
SQL> desc CVAR; NAME	NULL	TYPE

cvalue		real
SQL> desc CVEC; CVEC is NOT FINAL NAME	NULL	TYPE

ctimeVariant number(1)		
SQL> desc CVEC_TT; cvec_tt AS TABLE OF cvec CVEC is NOT FINAL NAME	NULL	TYPE

ctimeVariant number(1)		
SQL> desc DPRO; DPRO extends METH2.PMODEL DPRO is NOT FINAL NAME	NULL	TYPE

dpequation varchar(30)		
SQL> desc LCPRO ; LCPRO extends METH2.CPRO NAME	NULL	TYPE

clinear NUMBER(1)		
SQL> desc LDPRO ; LDPRO extends METH2.DPRO NAME	NULL	TYPE

llinear NUMBER(1)		
SQL> desc NLDPRO ; NLDPRO extends METH2.DPRO NAME	NULL	TYPE

nllinear NUMBER(1)		
SQL> desc NLCPRO ; NLCPRO extends METH2.CPRO NAME	NULL	TYPE

nclinear
NUMBER(1)

SQL> desc MEASVAR;

NAME	NULL	TYPE
mvalue		real

SQL> desc MMODEL;
MMODEL is NOT FINAL

NAME	NULL	TYPE
mequation		varchar(30)
mlinear		number(1)
mtimeVariant		number(1)
mvalueType		varchar(15)
pm		procmeas

SQL> desc MMODEL;
mmodel_tt AS TABLE OF mmodel

NAME	NULL	TYPE
mequation		varchar(30)
mlinear		number(1)
mtimeVariant		number(1)
mvalueType		varchar(15)
pm		procmeas

SQL> desc MNOISE;
MNOISE is NOT FINAL

NAME	NULL	TYPE
mncol		REAL
mncol		REAL

SQL> desc MNOISE_TT;
mnoise_tt AS TABLE OF mnoise

NAME	NULL	TYPE
mncol		REAL
mncol		REAL

SQL> desc MSVAR ;

NAME	NULL	TYPE
mmean		real
mmvalue		real
mnoise		real

SQL> desc MSVAR_TT ;
msvar_tt AS TABLE OF msvar

NAME	NULL	TYPE
mmean		real
mmvalue		real

```

mnoise
real

SQL> desc MSVEC;
MSVEC is NOT FINAL
NAME
----- NULL
TYPE
-----
msvrow
msvcol
real,
real

SQL> desc MSVEC_TT;
msvec_tt AS TABLE OF msvec
MSVEC_TT is NOT FINAL
NAME
----- NULL
TYPE
-----
msvrow
msvcol
real
real

SQL> desc NLMEAS ;
NLMEAS extends METH2.MMODEL
NLMEAS is NOT FINAL
NAME
----- NULL
TYPE
-----
tlinear
number(1)

SQL> desc PMODEL ;
PMODEL is NOT FINAL
NAME
----- NULL
TYPE
-----
pequation
plinear
ptimeVariant
pvalueType
pm
varchar(30),
number(1),
number(1),
varchar(15),
procmeas

SQL> desc PMODEL_TT ;
PMODEL_TT is NOT FINAL
NAME
----- NULL
TYPE
-----
pequation
plinear
ptimeVariant
pvalueType
pm
varchar(30),
number(1),
number(1),
varchar(15),
procmeas

SQL> desc PNOISE;
PNOISE is NOT FINAL
NAME
----- NULL
TYPE
-----
pnrow
pncol
REAL
real

SQL> desc PNOISE_TT;
pnoise_tt AS TABLE OF pnoise
PNOISE is NOT FINAL
NAME
----- NULL
TYPE
-----
pnrow
pncol
REAL
real

```

```
SQL> desc PROCMEAS;
NAME                                     NULL      TYPE
-----
id                                       NULL      varchar(30)
```

```
SQL> desc CV_COMPONENT ;
CV_COMPONENT  is NOT FINAL
NAME                                     NULL      TYPE
-----
convec                                               cvec_tt,
contvar                                              cvar
```

```
SQL> desc MM_COMPONENT ;
MM_COMPONENT  is NOT FINAL
NAME                                     NULL      TYPE
-----
meamod                                               mmodel_tt,
meastvec                                             msvec_tt,
meanoi                                               mnoise_tt,
procesmo                                             Pm_component
```

```
SQL> desc MN_COMPONENT ;
MN_COMPONENT  is NOT FINAL
NAME                                     NULL      TYPE
-----
meanoi                                               mnoise_tt,
mvariable                                             msvar,
cv_com                                               cv_component,
mm_com                                               mm_component,
meas_com                                             msv_component,
pmod_com                                             pm_component,
pn_com                                               pn_component,
sva_com                                              sva_component
```

```
SQL> desc PM_COMPONENT ;
PM_COMPONENT  is NOT FINAL
NAME                                     NULL      TYPE
-----
promodel                                             pmodel_tt,
statevec                                             svec_tt,
pronoise                                             pnoise_tt,
convec                                               cvec_tt
```

```
SQL> desc SVA_COMPONENT ;
SVA_COMPONENT  is NOT FINAL
NAME                                     NULL      TYPE
-----
statevec                                             svec_tt,
statevar                                             svar
```

```
select object_name, object_type, status from user_objects where object_type ='TABLE';
```

```
OBJECT_NAME                                OBJECT_TYPE    STATUS
-----
KF_NT                                       TABLE        VALID
KALMAN_FILTER                             TABLE        VALID
```

```
SQL> desc KF_NT ;
```

NAME	NULL	TYPE
------	------	------

```
SQL> desc KALMAN_FILTER;
```

Name	Null	Type
MEANOI		MNOISE_TT()
MVARIABLE		MSVAR()
CV_COM		CV_COMPONENT()
MM_COM		MM_COMPONENT()
MEAS_COM		MSV_COMPONENT()
PMOD_COM		PM_COMPONENT()
PN_COM		PN_COMPONENT()
SVA_COM		SVA_COMPONENT()

REFERENCES

1. Sommerville, "Software Engineering", Addison-Wesley, 9th -2011, pg 129-131
2. M. Blaha and W. Premerlani, "Object-Oriented Modeling and Design for Database Applications", Prentice Hall, 1998
3. E. Naiburg and R. Maksimchuk, "*UML for Database Design*", Addison-Wesley, 2001
4. R. Elmasri and I. Navathe, "*Fundamental of Database Systems fourth Edition*", Addison-Wesley, 2004
5. ISO/IEC 19501, Information Technology – Open Distributed Processing, : Unified Modeling Language (UML) Version 1.4.2 , 2005
6. Eisenberg, et al., SQL:2003 Has Been Published, SIGMOD Record, Vol. 33, No. 1, March 2004
7. R. Chennamaneni, "Comparison and Evaluation of Methodologies for transforming UML class diagram models to object-relational database systems" ,May 2006. University of North Dakota, USA.
8. L. Li and X. Zhao, "UML Specification of Relational Database", Journal of Object Technology, Vol. 2, No. 5, Sept-Oct 2003, pp. 87-100.
9. P. Rob and C. Coronel, "Databas Systems – Design, Implementation, & Management Fourth Ed.", Thomson Learning, 2000

10. G.Shelly, T. Cashman, et al, “Discovering Computers 2006”, Thompson Learning, 2005
11. M. Lamari, “Towards an Automated Test Generation for the Verification of Model Transformations”, ACM, March 2007
12. E. Marcos, B. Vela, et al, “A Methodological Approach for Object-Relational Database Design using UML”, Springer-Verlag 2003
13. E. Grant, J. Whittle, et al, “Checking Program Synthesizer Input/Output”, 3rd OOPSLA Workshop on Domain-Specific Modeling – OOPSLA03, Anaheim, California, October 2003.
14. Marcos, E., B. Vela, and J.M. Caverio. *Extending UML for Object-Relational Database Design*. In *UML 2001*. 2001: Springer-Verlag Berlin Hielderberg.
15. Mok, W.Y., D. P. Paper. “On Transformations from UML Models to Object-Relational Databases” . In 34th Hawaii Interantional Conference on System Sciences. Island of Maui, Hawaii, USA., 2001
16. Grady, B., James, R., et al. ,“The Unified Modeling Language User Guide”, Addison-Wesley, 1999
17. Codd, E.F. "Further Normalization of the Data Base Relational Model." (Presented at Courant Computer Science Symposia Series 6, "Data Base Systems," New York City, May 24th-25th, 1971.) IBM Research Report RJ909 (August 31st, 1971). Republished in Randall J. Rustin (ed.), *Data Base Systems: Courant Computer Science Symposia Series 6*. Prentice-Hall, 1972.
18. D. Lorentz and et. Al, “ Oracle Database SQL Language Reference 1g Releases (11.1), Oracle, August 2010

19. R. Chennamaneni and E. S. Grant, "Comparison and Evaluation of Methodologies for Transforming UML Models to Object-Relational Databases" , MICS, 2004
20. E. S. Grant, R. Chennamaneni and H. Reza, "Towards analyzing UML class diagram models to object-relational database systems transformations", ACM, 2006